Application of Artificial Neural Networks and the Wavelet Transform for Pattern Recognition, Noise Reduction and Data Compression

PhD Thesis

By

Muhammad Din Choudhry

Department of Mathematics

Islamia University

Bahawalpur, PAKISTAN

March 7, 2000

DE MONTFORT
UNIVERSITY

LEICESTER · MILTON KEYNES
BEDFORD · LINCOLN

Supervised by

Professor JM Blackledge

Department of Mathematical Sciences

Faculty of Computing Sciences and Engineering

DeMontfort University, Leicester, UK.

Contents

Abstract

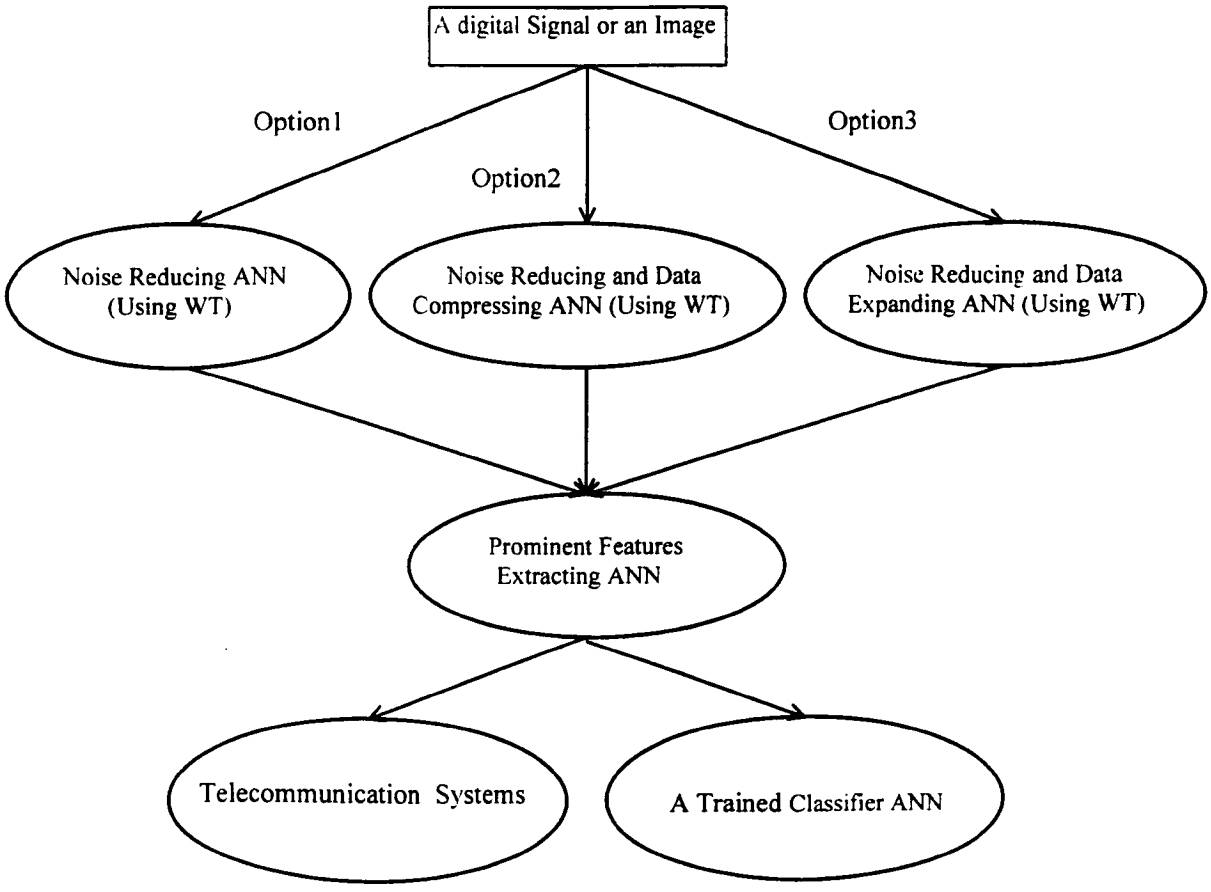
Theory of Artificial Neural Networks (ANNs) could not provide an exact method of weights training. The training is done mostly by iterative trial and error minimisation methods which do not enable the ANNs for time incremental learning. In this thesis, it is shown that the weights successfully produced by an error minimisation method are nothing more than the scaled versions of their respective components of the sample pattern and that the training methods leaves a chance for a neuron to be deceived. An exact method of weight construction is developed in the form of a system of linear equations. A new linear classifier ANN and a number of thresholding procedures are developed. It is proved that the Hopfield network and the Boltzmann machine do not qualify as the reasonable networks. A generalised multiclass linear classifier ANN is developed which is a combination of a newly developed multiclass linear ANN and a newly developed multiclass XOR classifier ANN. A biological neuromuscular system is interpreted as a multiclass linear classifier ANN. A new technique for pattern recognition, especially for images, has been presented with a software check. The technique minimises the design topology of ANNs and enables them to classify a scaled, a mirror image, and a noisy version of the sample pattern.

The Continuous Wavelet Transform (CWT), the Discrete Wavelet Transform, and the Wavelet Decomposition has been connected by developing an extend-able and intensify-able system of particular six Gaussian wavelets. A binary transform applicable for every real function is developed. The confusing automatic nature of the CWT is explained along with presenting a new style of defining wavelets. Application of the wavelet transforms for noise reduction and data compression/expansion is explained and their performance is checked through the self developed software. A modification in the CWT is made in order to make their application easier through ANNs. The ANNs are developed and their performance is checked against the self developed software. A new multiresolution zoom-out wavelet transform is developed which expands data without smoothing it. A new wavelet is deduced from the smoothing average filter. Some two-dimensional wavelets for noise reduction and data compression/expansion are developed on the same style and their performance is checked through the self developed software. An ANN for CWT using a newly developed two-dimensional wavelet is developed and its activation is explained. Data compression by locating peaks and bottoms of data and setting other elements equals zero is done with the guarantee of reconstruction. The new wavelet transform is modified to reconstruct the data between peaks and bottoms. Peaks and bottoms detecting ANNs are developed and their performance is checked against the self developed software. Procedures for classification are presented with self developed software check. The theory of ANNs requires bit-wise parallel adders and multiplexors. A parallel adder circuit is developed by combining some newly developed basic units for the purpose.

Objectives of Research

The objectives of this research is to develop some logical structures for a digital computer system that can work like human brain. This study comes in the field of Artificial Neural Network(ANN). Parallel processing and wisdom being the two main properties of human brain, will be maintained throughout the research. That is, each component of the desired computer system should consist of a parallel processor and should be wise enough to perform a specified job accurately without requiring any software and any sequential micro processor. As the human brain is to perform a large number of jobs and activities, so research on this issue may be continued for years.

However, this PhD thesis is aimed to produce ANNs whose logic and design can lead us to produce, at least, a Digital Computer for Image Processing and Pattern Recognition. A basic proposed structure of which is given in the following;



INTRODUCTION

Part 1

1.1 Chapter1: Chapter-wise introduction is presented in this chapter.

1.2 Chapter2: Rejection of Old and Development of New Procedures for Artificial Neural Networks

In Sections(2.1-2), error minimisation methods of weight training are rejected by showing that a trained neuron is deceive-able and that weights are nothing more than a scaled version of components of the sample pattern. In Section(2.3), a new weights construction method is introduced which, unlike the old methods, provides exact and unique weights. The Section(2.4) through Section(2.19) provides the procedure of a straight line activation, a class of objects and its thresholds, linear activation and classification of classes, combining the high force and the low force thresholding, image data and extracting contours, finding individual thresholds for a class of objects, from individual thresholds to segment thresholds, finding weights and time incremental learning, developing binary transform and its inverse. Some basics follow:

1.2.1 Parallel Processing: Idea of parallel processing can be visualised from the ANN shown in Figure(1.1). The ANN performs the task of Moving Average Scheme.

$$y_0 = x_0, \quad y_i = \frac{x_{i-1} + x_i}{2}, \quad i = 1,2,\ldots,n$$

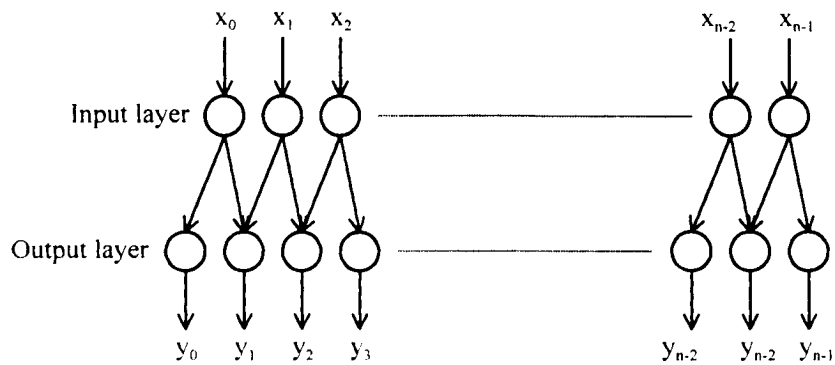


Figure1.2.1 An ANN to perform the Moving Average Scheme.

All inputs are passed simultaneously to input layer which is non-processing.

1.2.2 Training: Objective of training of a neuron is to find $w_i's$ which can satisfy the equation

$$(W, X^s) = \sum_i w_i x_i^s = TI$$

where $X^s = (x_0^s, x_1^s, ..., x_{n-1}^s)$, $W = (w_0, w_1, ..., w_{n-1})$, and TI represent a sample signal, a weights signal, and the given target input respectively.

1.2.3 Time incremental Learning: The capability of future learning for additional samples without requiring those for which the ANN has learnt previously is called *time incremental learning*. For example, a person, who is trained to drive a bicycle, requires only additional controls to learn for driving a motorcycle.

1.2.4 Activation: Activation of a trained neuron is given, for example, as

$$net = \sum_i w_i x_i^a \; ; \quad out = f(net) = \begin{cases} 1, & if\, net \geq TI \\ 0, & if\, net = TI \\ -1, & if\, net \leq TI \end{cases}, \quad i = 1,2,...,n$$

where x_i^a are the components of an actual pattern X^a.

1.2.5 Supervision: A neuron is supervised to produce its output, for example, as:

upgrade weights to produce $out = 1$ when we present a sample from class A

upgrade weights to produce $out = 0$ when we present a sample from class B

1.2.6 Hebbian Learning Rule: Produce target output (correct output) by upgrading $w_i's$ according to the following rule ;

$$\text{if correct, } w_i(t+1) = w_i(t)$$
$$\text{if } out \text{ is 0, should be 1 (class A), } w_i(t+1) = w_i(t) + x_i^s$$
$$\text{if } out \text{ is 1, should be 0 (class B), } w_i(t+1) = w_i(t) - x_i^s$$

Gain term: A constant η, $0 < \eta \leq 1$, called the gain term, is required to slowdown the weights up-gradation so that the previous learning may not be spoiled.

$$\text{if } out \text{ is 0, should be 1 (class A), } w_i(t+1) := w_i(t) + \eta x_i^s$$
$$\text{if } out \text{ is 1, should be 0 (class B), } w_i(t+1) := w_i(t) - \eta x_i^s$$

1.2.7 Windrow-Hoff Delta Rule: This is the up-graded Hebbian learning rule. Let x_i^s and w_i^s denotes the *ith* component of the sample pattern X^s and its corresponding weight respectively, then the weights training by the delta rule is done as:

Weights training: If *dout* and *aout* denote the desired and the actual output, respectively, then

$w_i(0) = x_i^s$
$aout = f(net)$
$\Delta = dout(0) - aout(0)$
$t = 1$
$while(\Delta != 0)\{$
$\quad w_i(t) = w_i(t-1) + \eta \Delta x_i^s$
$\quad aout = f(net)$
$\quad \Delta = dout(t) - aout(t)$
$\quad t = t + 1$
$\}$

1.2.8 Linear Activation: From the properties of a straight line, we know that

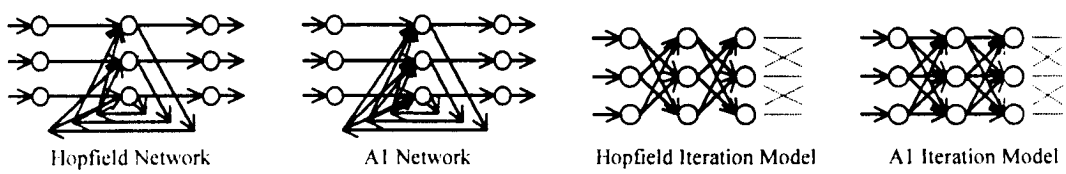
$$x_2 + mx_1 + c \begin{cases} > 0, & \text{if the point } (x_1, x_2) \text{ lies above the line.} \\ = 0, & \text{if the point } (x_1, x_2) \text{ lies on the line.} \\ < 0, & \text{if the point } (x_1, x_2) \text{ lies below the line.} \end{cases}$$

Observe that

$$net = w_1 x_1 + w_2 x_2 + w_0 TI = x_2 + mx_1 + c = 0 \Rightarrow m = w_1 / w_2 \text{ and } c = w_0 TI / w_2 .$$

1.3 Chapter3: Rejection of Old and Development of New Classifier Artificial Neural Networks

In the first three sections of this chapter, it is proved by presenting a series of new ANNs that the weights of the Hopfield network and that of the Boltzmann Machine are either destructive or redundant. Moreover, its is proved that both the Hopfield network and the Boltzmann Machine do not qualify as a reasonable network. Further, that the weights of Boltzmann Machine are shifted versions of the weights of Hopfield network.



In Section(3.4), a new classifier ANN is presented which classifies m linearly inseparable classes without using a non-linear activation function and at the same time it provides a demonstration to adopt the new weight training method, linear activation, the high force and the low force thresholding. The design topology of the ANN shows that a scaled version, whether it is noisy or mirror image, of the sample pattern can be classified by using some suitably defined activation functions. In Section(3.5), A multiclass ANN acted upon segment thresholds is developed. The ANN demonstrates reducing of design topology of classifier ANNs by using a straight line activation and the high force and low force segment thresholds. An XOR ANN for two objects, two XOR ANNs for three objects, two XOR ANNs for n objects have been newly developed in Section(3.6). The ANNs are straight line activated. In Section(3.7), a training procedure of an XOR ANN is presented. In Section(3.8) and Section(3.9), a new XOR ANN and its generalised XOR ANN detector for m objects are presented.

1.4 Chapter4: Biological Neuromuscular Systems and Artificial Neural Network

In a motor unit, (consisting of a neuron, an axon, terminal branches of the axon, and all the muscle fibres supplied by these branches, see Figure(4.1-2)), when the neuron approaches the muscle, by sending an electrical potential V along the axon, a contraction of muscle fibres produces kinetic and potential energies say K and P respectively. If f denotes the fractional force on the muscle fibres due to internal and external interference, then we can write

$$V = K + P - f = T \ ,$$

where T denotes the net work done by the muscle.

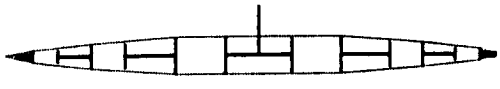


Figure 1.4.1: A single Muscle fibber at rest.

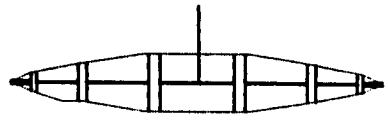


Figure 1.4.2: A Muscle fibber contracted.

From artificial neural network point of view, this work done can be interpreted as the target output demanded by the neuron from the muscle. The lowest and the greatest contraction limits, say α_i and β_i of the *ith* muscle fibber x_i can be used to compute Γ_i

and τ_i. the high force threshold value and the low force threshold value for an artificial neuron, as

$$\Gamma_i = T\alpha_i^2 / \sum \alpha_i^2 = u_i \alpha_i, \text{ where } u_i = T\alpha_i / \sum \alpha_i^2$$

$$\tau_i = T\beta_i^2 / \sum \beta_i^2 = v_i \beta_i, \text{ where } v_i = T\beta_i / \sum \beta\alpha_i^2$$

So that the activations of the hidden layer of the artificial neural network, on receiving an actual pattern $X = (x_1, x_2, ..., x_n)$, become

$$net^h = \sum_{i=1}^{n} \Gamma_i - T = \sum_{i=1}^{n} u_i x_i - T \; ; \quad f_1(net^h) = \begin{cases} 1, \text{ if } net^h \le 0 \\ 0, \text{ otherwise} \end{cases}$$

$$net^l = \sum_{i=1}^{n} \gamma_i - T = \sum_{i=1}^{n} v_i x_i - T; \quad f_1(net^l) = \begin{cases} 1, \text{ if } net^l \ge 0 \\ 0, \text{ otherwise} \end{cases}$$

where a requirement T within the limits $T_{\max} = \sum_{i=1}^{n} u_i \alpha_i$ and $T_{\min} = \sum_{i=1}^{n} v_i \beta_i$ is fulfilled by the muscle as shown in Figure(1.4.3).

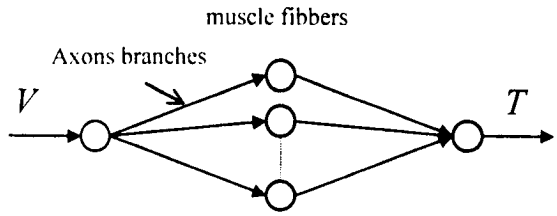


Figure 1.4.3: Analogue Neuromuscular Network

And the activation of the output neuron is

$$net = f_1(net^h) + f_1(net^l); \quad f_2(net) = \begin{cases} 1, \text{ if } net = 2 \\ 0, \text{ otherwise} \end{cases}$$

For linear activation see chapters(2-3).

1.5 Chapter5: Artificial Neural Networks for Image Processing

Peaks and bottoms of an image data look like tops of hills and depths of valleys. Locating peaks and bottoms, their amplitudes, their indices or/and their relative distances is found advantageous for classification and telecommunication of signals/images. Variation of features in different situations like; a still Image and its features, luminance and variation of features, telescopic image and variation of features etc. requires to investigate and elaborate their possible situations. This is one of the issues of this chapter.

Further, to minimise design topology of ANNs, data reduction is required. Data reduction can be done by picking amplitudes and indices of peaks and bottoms and discarding other elements data. It is found (and will be shown with software check in Chapter10) that the amplitudes and indices of peaks and bottoms are necessary and sufficient features to classify and reconstruct the signals/images. A basic structure of a classifier ANN is developed in Section(5.4). Its advanced model, with training and topological description, is presented in Section(5.6-7). The design topology of the ANN makes it capable of classifying a darker, a brighter, a smaller, and a larger version of the image with suitably defined linear activating functions. In Section(5.8), a remarkable classifier ANN is presented with software check. The classifier is capable of detecting and reconstructing an input pattern as a known or an unknown one. In later case, the pattern is further classified into three categories; a scaled version, or a mirror image, or a noisy version of the sample pattern. Moreover, the ANN is capable to tell us whether the noise is removable or not. This ANN can be considered as a good example in pattern classification.

Part 2

1.6 Chapter 6: From Fourier Analysis to Wavelet Analysis

A number of development has been made in this chapter. In Sections(6.1-6.5), the Fourier transform and its drawbacks, the Short Time Fourier Transform (STFT) and its computing redundancy, and the relative frequency analysis are elaborated. The facts are tested and demonstrated by developing software programs with graphical outputs. In Section(6.7-8), a wavelet transform is achieved from the STFT and its performance is checked by developing software having graphical outputs. In Section(6.9-10), wavelet decomposition is described mathematically with an example algorithm. An important development is made in Section(6.11). It is shown that a real time signal can be decomposed in terms of particular six Gaussian Wavelets. There are six programs developed and their graphical outputs are included in the text. The Section(6.12) has the honour to associate two wavelet transforms, the DWT and the CWT, with the wavelet decomposition introduced in Section(6.11). In Section(6.13) some larger wavelet packets

are developed by utilising the Gaussian wavelet packet of six windows. In Section(6.14), the Gaussian wavelet packet is transformed to a binary wavelet packet which provides us orthonormal wavelet basis while preserving its application for digitised CWT for noise reduction and data compression. In Section(6.15-17) some useful refinement relations has been interpreted and elaborated mathematically. The developments are presented with graphical support. Basics of the theory are given below:

1.6.1 Short Time Fourier Transform

Let $L^2(0,2\pi)$ denotes the space of 2π-periodic square-integrable functions. The Fourier transform (FT) and IFT of a function $f(t) \in L^2(0,2\pi)$ are defined, respectively, as

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt \quad \text{and} \quad f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(\omega)e^{i\omega t}d\omega .$$

The Short Time Fourier Transform (STFT), denoted by $STFT_f(p,q)$, of a signal $f(t)$ is defined as:

$$STFT_f(p,q) = \int_{-\infty}^{\infty} f(t)w(t - p\tau_0)e^{-iq\omega_0 t}dt$$

where $w(t)$ is a window function and p, q specifies the time and frequency dimension, respectively[2]. Its digitised version (see Section(8.1) for details) is given below:

$$STFT_f(p,q) = \sum_{i=0}^{l-1} f(i)w(i + p)e^{-iq\omega_0 i}, \text{ where } p = 0,1,\ldots,(l - m)$$

where l and m denote the signal length and the window width respectively. It has been shown in Section(6.3) that the STFT has a computing redundancy which can be removed by shifting the translation parameter from window to the input function to have

$$STFT_f(p,q) = \sum_{i=0}^{m-1} f(i + p)w(i)e^{-iq\omega_0 i}, \text{ where } p = 0,1,\ldots,(l - m)$$

1.6.2 STFT leading to Continuous Wavelet Transform

According to [9], the scaled version of the wavelet $\psi(t) = w(t)e^{-i2\pi\omega_0 t}$ can be written as

$$\psi_a(t) = \frac{1}{\sqrt{|a|}} \psi(\frac{t}{a})$$

where a is the scale factor (that is $\omega = \frac{\omega_0}{a}$), and the constant $\frac{1}{\sqrt{|a|}}$ is for energy normalisation. The STFT then becomes

$$STFT(\tau,a) = \int_{-\infty}^{\infty} f(t)\psi(\frac{t-\tau}{a})dt .$$

This is called as Wavelet Transform, where $\psi(t)$ is called the basic wavelet. Changing notation, we have Wavelet Transform (WT) and Inverse Wavelet Transform (IWT) given as

$$WT(\tau,a) = \int_{-\infty}^{\infty} f(t)\psi(\frac{t-\tau}{a})dt \quad \text{and} \quad f(t) = c\int_{-\infty}^{\infty} WT(\tau,a)\psi(\frac{t-\tau}{a})dt$$

respectively.

1.6.3 Wavelet Basis and Discrete Wavelet Transform

Definition 1: It is stated in [6] that the norm for the space $L^2(\Re)$ is defined as

$$\|f\|_2 = \langle f,f\rangle^{1/2} .$$

where the inner product of $f,g \in L^2(\Re)$ is given as $\langle f,g\rangle = \int_{-\infty}^{\infty} f(x)\overline{g(x)}dx$

If ψ $L^2(\Re)$ is a wavelet basis, then so is the every $\psi_{j,k} = 2^{j/2}\psi(2^j x - k)$, $j,k \in Z$. That is, $||\psi_{j,k}||=||\psi||$ for all $j,k \in Z$. If a function ψ $L^2(\Re)$ has a unit length, then each of the functions $\psi_{j,k}$ has also unit length.

Definition 2: A function ψ $L^2(\Re)$ is called an orthonormal wavelet (o.n. wavelet), if the family $\{\psi_{j,k} = 2^{j/2}\psi(2^j x - k),\ j,k \in Z\}$ is an orthonormal basis (wavelet basis) of $L^2(\Re)$; That is, if

$$\langle \psi_{j,k}, \psi_{l,m}\rangle = \delta_{j,l}.\delta_{k,m},\ j,k,l,m \in Z.$$

Definition 3: Let $f \in L^2(\Re)$ and $\{\psi_{j,k}\ j,k \in Z\}$ is a wavelet basis then f can be expressed as

$$f(x) = \sum_{j,k=-\infty}^{\infty} c_{j,k}\psi_{j,k}(x)$$

where constants $c_{j,k}$, called wavelet coefficients, are given by

$$c_{j,k} = \sum_{j,k=-\infty}^{\infty} f(x)\psi_{j,k}(x)$$

and the expression $f(x) = \sum_{j,k=-\infty}^{\infty} c_{j,k}\psi_{j,k}(x)$ itself is called a Discrete Wavelet Transform (wavelet series) of the function f.

1.6.4 Wavelet Decomposition and Refinement Relations

Definition 1: It is stated in [6] that any $f \in L^2(\Re)$ can be approximated, as closely as is desired, by an $f_n \in V_n$, for some $n \in Z$. For, $V_n = V_{n-1} \oplus W_{n-1}$ implies that f_n can, uniquely, be decomposed as:

$$f_n = f_{n-1} + g_{n-1}, \text{ where } f_{n-1} \in V_{n-1} \text{ and } g_{j-1} \in W_{n-1}.$$

By repeating this process, we have

$$f_n = g_{n-1} + g_{n-2} + \ldots + g_{n-m} + f_{n-m}$$

This decomposition is called *wavelet decomposition.*

Definition 2: Since both the scaling function $\phi \in V_0$ and the wavelet function $\psi \in W_0$ are in V_1, and since V_1 is generated by $\phi_{i,k}(x) = 2^{1/2}\phi(2x - k)$, $k \in Z$, there exist two sequences $\{c_k\}$ and $\{d_k\} \in l^2(\Re)$ such that

$$\phi(x) = \sum_k c_k \phi(2x - k)$$

$$\psi(x) = \sum_k d_k \phi(2x - k)$$

for all $x \in \Re$. These formulas are called the ***refinement or dilation equations or two-scale relations*** of the scaling function and wavelet, respectively[6]. On the other hand, since both $\phi(2x)$ and $\phi(2x-1)$ are in V_1 and $V_1 = V_0 \oplus W_0$, there are four l^2-sequences which we denote by $\{a_{-2k}\}$, $\{b_{-2k}\}$, $\{a_{1-2k}\}$, and $\{b_{1-2k}\}$, $k \in Z$ such that for all $x \in \Re$;

$$\phi(2x) = \sum_k \left[a_{-2k}\phi(x - k) + b_{-2k}\psi(x - k)\right];$$

$$\phi(2x-1) = \sum_k \left[a_{1-2k}\phi(x - k) + b_{1-2k}\psi(x - k)\right],$$

Combining the two formulas into a single, we have

$$\phi(2x-l)=\sum_{k}\left[a_{l-2k}\phi(x-k)+b_{l-2k}\psi(x-k)\right].$$

which is called ***decomposition relation*** of ϕ and ψ.

Now we have two pairs of sequences $\left(\{c_k\},\{d_k\}\right)$, called reconstruction sequences, and $\left(\{a_k\},\{b_k\}\right)$, called decomposition sequences; all of which are unique due to the direct sum relationship $V_1 = V_0 \oplus W_0$. These sequences are used to formulate the reconstruction and decomposition algorithms[6].

Definition 3: To downsample a sequence, we simply keep its every other term. More precisely, in the following example of downsampling, only the terms with even indices are kept, and the (even) indices of the output sequence are halved. If we denote by $\tilde{\mathbf{c}}^j$ the downsampled version of the sequence $\mathbf{c}^j = \left\{c_l^j,\ l \in Z\right\}$, then $\tilde{\mathbf{c}}^j$ is given as follows:

$$\begin{cases}\tilde{\mathbf{c}}^j = \left\{\tilde{c}_l^j,\ l \in Z\right\};\\ \text{with}\\ \tilde{c}_l^j := c_{2l}^j.\end{cases}$$

Definition 4: To upsample a sequence, a zero is placed in between every two consecutive terms of the sequences. More precisely, in the following example of upsampling, the indices of the input sequences $\mathbf{c}^{j-1} = \left\{c_l^{j-1},\ l \in Z\right\}$ and $\mathbf{d}^{j-1} = \left\{d_l^{j-1},\ l \in Z\right\}$ are multiplied by 2, and zeros are used at the odd indices.

If we denote by $\tilde{\mathbf{c}}^j$ the downsampled version of the sequence $\mathbf{c}^j = \left\{c_l^j,\ l \in Z\right\}$ and, respectively, then $\tilde{\mathbf{c}}^j$ is given as follows:

$$\begin{cases}\tilde{\mathbf{c}}^j = \left\{\tilde{c}_l^j,\ l \in Z\right\};\\ \text{with}\\ \tilde{c}_{2k}^j := c_k^j,\ \text{and}\\ \tilde{c}_{2k+1} := 0,\ k \in Z\end{cases}$$

1.6.5 Associating Wavelet Transforms with the Wavelet Decomposition

A system of six relative bandwidth Gaussian windows is developed in terms of which a real function $f(x)$ can be decomposed as

$$f(x) = \sum_{n} g_n(x) f(x) + err(x)$$

The functions $g_n(x)$ are defined by

$$g_j(x) = \frac{1}{2\sqrt{\pi 2^j}} \exp(-(x - cj)^2 / 2^{j+2}), \quad j = 1,2,...6.$$

where c is an integre in $[0, j]$ and window $width = 2^j$ $(= 2\sqrt{\alpha}\ ,\ \alpha = 2^{2j-2}\ )$.

After multiplying with the constant factor 2 and the relative normalising factor $width$ equals 2^k, we have

$$g_k(x) = \frac{2^k}{\sqrt{\pi 2^k}} \exp(-(x - ck)^2 / 2^{k+2}), \quad k = 1,2,...6$$

where c is an integre in [0.k] and window $width = 2^k$ $(= 2\sqrt{\alpha}\ ,\ \alpha = 2^{2k-2}\ )$.

We can visualise the given system $\{g_k(i)\}_{i=1}^{n}$, $(k = 1,2,.....6)$ as a k x n matrix. Then the function

$$\{\psi_i(k)\}_{i=1}^{6} = \frac{1}{\lambda_k}\{g_k(i)\}_{i=1}^{n}, \quad k = 1,2,...6$$

represents the normalised kth column of the matrix. If $f(x)$ is a real signal to be transformed (decomposed), then for each fixed k we have

$$\overset{w}{f}_k = \sum_{i=1}^{6} \psi_i(k) f(k) \tag{1.6.5a}$$

$$\sum_{i'}^{6} \psi_{i'}(k) \overset{w}{f}_k = \sum_{i'}^{6} \psi_{i'}(k) \left\{ \sum_{i=1}^{6} \psi_i(k) f(k) \right\} = \sum_{i,i'=1}^{6} f(k) \psi_i(k) \psi_{i'}(k) = \sum_{i,i'=1}^{6} f(k) \delta_{i,i'}$$

For $i' = i$, we have

$$f(k) = \sum_{i=1}^{6} \psi_i(k) \overset{w}{f}_k \tag{1.6.5b}$$

Equation(1.6.5a) and Equation(1.6.5b) are called Discrete Wavelet Transform (DWT) and the Discrete Inverse Wavelet Transform (DIWT) respectively. Where the set $\{\psi_i(k)\}_{i=1}^{6}$ forms an orthonormal basis. An other important wavelet transform, called the Continuous Wavelet Transform, along the rows of the matrix, which should also be associated with the wavelet decomposition. We will express it along with its application to noise

reduction and data compression in Chapters(7-9) in detail, however, we can define it immediately as:

$$f_j(k) = \frac{2^{j/2}}{\lambda}\sum_{i=1}^{l} f(i-1)\psi(2^j i - k)\,, \text{ where } k = 0.1.....2^j(l-m)$$

where $[0,l[$ denotes the domain of both the input function f and the wavelet function ψ and m ($m < l$) be the number of non-zero elements of ψ that constitute a window.

1.6.6 A Binary Wavelet Packet and Discrete Wavelet Transform

We achieve binary wavelet packet, denoted by $\left\{g_k^b(i)\right\}_{i=1}^{n}$, from the system $\left\{g_k(i)\right\}_{i=1}^{n}$ by applying a three stage hard transforming function given in Equation(6.14.1). This system is important for both the wavelet transform and the pattern recognition. The binary coded form of $\left\{\psi_i(k)\right\}_{i=1}^{6}$ is achieved as

$$\left\{\psi_i^b(k)\right\}_{i=1}^{6} = \frac{1}{\lambda_k^b}\left\{g_k^b(i)\right\}_{i=1}^{n}\,, \quad k = 1,2,.....,6.$$

Similarly, by replacing $\left\{\psi_i(k)\right\}_{i=1}^{6}$ with $\left\{\psi_i^b(k)\right\}_{i=1}^{6}$ in Equation(1.6.5a) and Equation(1.6.5b), we have the required binary wavelet transform of a real signal $f(x)$ and its inverse transform respectively as

$$\overset{w}{f}_k = \sum_{i=1}^{6}\psi_i^b(k)f(k) \qquad (1.6.6a)$$

$$f(k) = \sum_{i=1}^{6}\overset{w}{f}_k\,\psi_i^b(k) \qquad (1.6.6b)$$

1.6.7 The Binary Wavelet Packet and Continuous Wavelet Transform

As mentioned earlier, the second wavelet transform associated with the wavelet decomposition is the noise reducing plus data compressing Continuous Wavelet Transform. For a single wavelet g^b the digitised version of continuous wavelet is defined as (see details in Chapter7-8) given below:

$$f_j(k) = \frac{2^{j/2}}{\lambda}\sum_{i=1}^{l} f(i-1)g^h(2^j i - k), \text{ where } k = 0.1.....2^j(l-m) \qquad (1.6.7a)$$

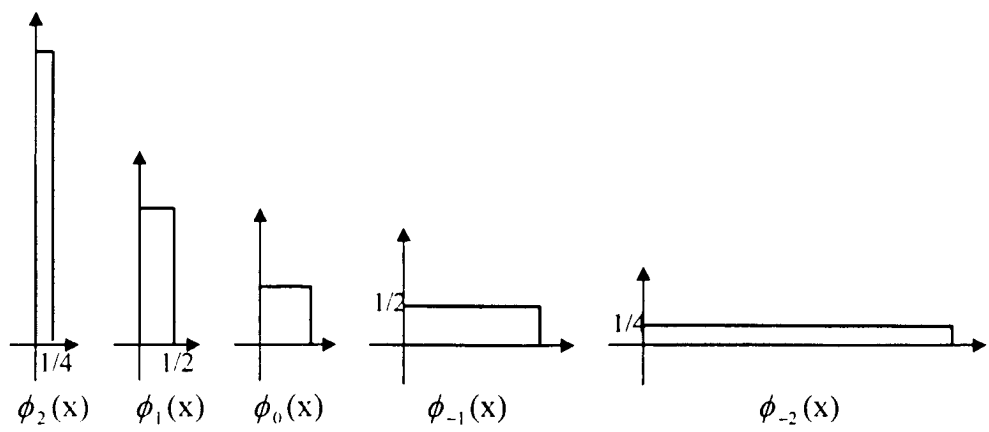
where the input real function f is defined on $[0,l[$, and m $(m < l)$ is the width of the wavelet g^h. If $\{g_r^h(i)\}_{i=1}^{n}$, $(r = 1.2,....,6)$ denotes the binary wavelet packet, then in view of Equation(1.6.7a), we have the continuous wavelet transform as

$$f_j(k) = \frac{2^{j/2}}{\lambda}\sum_{r=1}^{6}\sum_{i=1}^{l} f(i-1)g_r^h(2^j i - k), \quad k = 0,1,....,2^j(l - wwp),$$

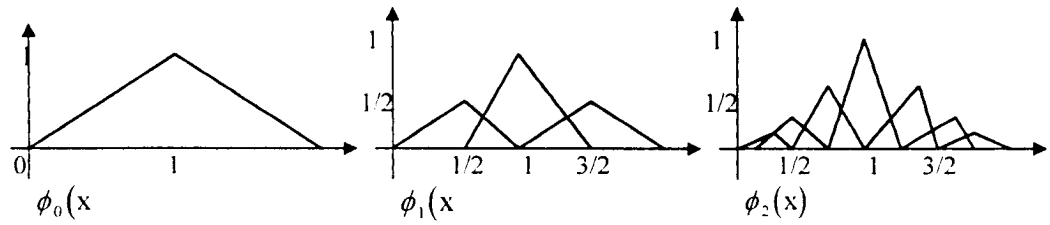
where wwp denotes width of the wavelet packet.

1.6.8 Some Mathematical Developments

Section(6.15) provides two different ways of mathematically developing the relative bandwidth system of wavelets shown below:



The fact that the *high pass filtering* a signal keeps its scale but increases its resolution is shown mathematically by using a refinement equation. This development is also an example that a wavelet defined on the style of Equation(7.2.2) can not expand automatically.



1.7 Chapter 7: Continuous Wavelet Transform and its Application

In Section(7.1), the Continuous Wavelet Transform (CWT) is elaborated and its digitised version is expressed. A confusing difference between the expansion of continuous wavelets and digital wavelets is focused in detail in Section(7.2) with the help of a new wavelet introduced there. In Sections(7.3-7.5), both the magical behaviour of zoom-out plus noise reducing wavelet transform of decreasing the wavelet window-width automatically on the increase of data expansion and the behaviour of zoom-in plus noise reducing wavelet transform of increasing the wavelet window-width automatically on the increase of data compression are demonstrated and performance of the transforms is checked against the software developed for this purpose. In Section(7.6), A multi-option ANN for noise reduction and data compression is presented.

1.7.1 Some Basics of Continuous Wavelet Transform

It is expressed in [8] that the continuous wavelet transform can be written as

$$CWT_f(b,a) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(x)\psi(\frac{x-b}{a})dx .$$

where $a > 0$ and b are called the scaling and the shift parameters respectively.

Let $[0,l[$ denote the domain of definition of f and ψ and let m, $m < l$, be the number of non-zero elements of ψ that constitute a window, then we have

$$CWT_f(k,j) = \frac{1}{\lambda} \int_{2^{-j}k}^{2^{-j}k+m-1} f(x)\psi(2^j x - k)dx \text{ , where } k = 0,1,....,2^j(l-m) .$$

And its digitised version is

$$f_j(k) = \frac{2^{j/2}}{\lambda} \sum_{i=1}^{l} f(i-1)\psi(2^j i - k) \text{, where } k = 0,1,.....2^j(l-m)$$

It can be visualised that; (i) ψ is compressed by a factor of $1/2^j$, $j > 0$, moves along the real axis with step-size equals 1 and produces 2^j elements in each step and the transform is called zoom-out wavelet transform, (ii) ψ is expanded by a factor 2^j, $j > 0$, moves along the real axis with step-size equals 2^j and produces 1 element is each step and the transform is called zoom-in wavelet transform. This happens automatically depending

upon the given value of j. The automatic nature, which is blessing, is found tedious to visualise. Readers are suggested to study Section(7.2).

1.7.2 Orthogonality of Wavelets

A wavelet function ψ is called an orthonormal wavelet if its family, denoted by $\{\psi_{j,k} = 2^{j/2}\psi(2^j x - k),\quad j,k \in Z\}$, form a system of orthonormal wavelets; that is, if

$$\langle \psi_{j,k}, \psi_{l,m} \rangle = \delta_{j,l}.\delta_{k,m},\quad j,k,l,m \in Z.$$

It has been observed practically that the mutual scalar product of different family members is not required in the application of CWT for data compression and noise reduction. However, the family of wavelets should be orthogonal. That is, the family members should satisfy

$$\langle \psi_{j,k}, \psi_{l,m} \rangle = 1, \text{ when } j = l \text{ and } k = m.$$

Let us consider, for example, a basic wavelet ψ defined as

$$\{\psi(i)\}_{i \in Z} = \frac{1}{\sqrt{60}}[\ldots,0,1,2,3,4,4,3,2,1,0,\ldots].$$

For j=0, we have two adjacent family members

$$\psi_{0,k} = \frac{1}{\sqrt{60}}[\ldots,0,1,2,3,4,4,3,2,1,0,\ldots],\quad \psi_{0,k+1} = \frac{1}{\sqrt{60}}[\ldots,0,0,1,2,3,4,4,3,2,1,0,\ldots]$$

which are orthogonal since $\langle \psi_{0,k}, \psi_{0,k} \rangle = \langle \psi_{0,k+1}, \psi_{0,k+1} \rangle = 1$ but are not arthonormal since $\langle \psi_{0,k}, \psi_{0,k+1} \rangle \neq 0$. The family members of a wavelet can be used as it were orthonormal by avoiding to compute their mutual scalar product throughout the application.

1.7.3 A New Wavelet with a New Defining Style and the Automatic Nature of CWT

For $t \in \Re$ and $m \in Z^+$, we can define a basic wavelet window function ψ as

$$\psi(t) = \frac{1}{\lambda}\begin{cases} 0, & t \leq 0 \\ t, & 0 < t \leq m/2 \\ m - t, & m/2 < t < m \\ 0, & t \geq m \end{cases}$$

where λ denotes the energy of ψ. The concept of automatic wavelet compression/expansion during the wavelet transform is concern with its family members.

1.7.4 Automatic Nature of Wavelets

In order to visualise the automatic nature of wavelets and to remove its confusion, it is found necessary to define its family members in two ways; one for continuous and the other for digital wavelets.

Case-1: A continuous family of the basic wavelet is defined as

$$\psi(2^j t) = \frac{1}{\lambda}\begin{cases} 0, & t \le 0 \\ 2^j t, & 0 < t \le m/2 \\ 2^j(m-t) & m/2 < t < m \\ 0, & t \ge m \end{cases}$$

This style of definition is suitable for zoom-out plus noise reduction wavelet transform. See Section(7.4) for details. But this is miss-leading for automatic wavelet expansion during zoom-in plus noise reduction wavelet transform. Hence we are lead to a new style of defining the wavelets as given below in case-2.

Case-2: A digital family of a basic wavelet for zoom-in wavelet transform should be defined as

$$\psi(2^j t) = \frac{1}{\lambda}\begin{cases} 0, & 2^j t \le 0 \\ t, & 0 < 2^{-j} t \le m/2 \\ m-t & m/2 < 2^j t < m \\ 0, & 2^j t \ge m \end{cases} = \frac{1}{\lambda}\begin{cases} 0, & t \le 0 \\ t, & 0 < t \le m/2^{j+1} \\ m-t & m/2^{j+1} < t < m/2^j \\ 0, & t \ge m/2^j \end{cases}$$

This definition is blessing for zoom-in plus noise reduction transform but is miss-leading for zoom-out plus noise reduction. In order to have width equals integral power of 2, we define the family of ψ slightly different as given below:

$$\psi(2^j t) = \frac{1}{\lambda}\begin{cases} 0, & t \le 0 \\ t, & 0 < t \le m/2^{j+1} \\ m-(t-1) & m/2^{j+1} < t \le m/2^j \\ 0, & t \ge m/2^j \end{cases}$$

1.8 Chapter8: Continuous Wavelet Transform and its Modification

The digitised version of the Continuous Wavelet Transform presented in Chapter7 is modified in this Chapter8. The shift and the resolution parameters are shifted from the wavelet to the signal to be transformed. As a result of this modification the variation of the resolution parameter unlike that in Chapter7 does not changes the window width.

The application of the modified wavelet transform has been explained in order to have (i) an ANN of wavelet transform for noise reduction, (ii) an ANN of wavelet transform for noise reduction plus data compression, (iii) an ANN of wavelet transform for noise reduction plus data expansion, and (iv) an ANN of wavelet transform for data expansion without noise reduction. At the end of the Chapter two feedback ANNs for noise reduction and data compression has been presented.

1.8.1 A Modification in CWT

The automatic nature of CWT may be complicated to implement. In order to avoid such complications we shift the zoom-in and zoom-out characteristics from the wavelet ψ to the input function f to have

$$CWT_f(k,j) = \frac{1}{\lambda}\int_0^{m-1} f(x+2^j k)\psi(x)dx\,,\quad k = 0,1,....,l/2^j \qquad (1.8.1a)$$

This modified CWT expresses that the $(l-m)/2^j+1$ successive segments overlapped by $m-2^j$ elements of the function f are passed with step-size equals 2^j for noise reduction in front of the wavelet ψ consisting of only m elements which are non-zero and constitute the window.

1.8.2 Digitisation: Equations(1.8.1a) can be digitised as

$$f_j(k) = \frac{1}{\lambda}\sum_{i=0}^{m-1} f(i+2^j k)\psi(i)\,,\text{ where } k = 0,1,...,l/2^j \qquad (1.8.1b)$$

Digitised signals and images consist of values (elements) rather than the definitions of their generating functions. But the application of the transform given in Equations(1.8.1b) for data expansion which corresponds to the negative values of j, the presence of the term $k/2^j$ in $f(i+k/2^j)$ demands us unavailable values in between each two consecutive elements of the signal f. In other words, the signal f is treated as it were a

continuous one. An easy fix to this problem lies in dropping the factor $1/2^j$ from $f(i+k/2^j)$ to have $f(i+k)$. However, in order to remove this drawback and to have a new zoom-out wavelet transform that can expand data without smoothing the original data, we can further modify the transform given in Equation(1.8.1b).

1.8.3 Further Modification

If $Q(\frac{k}{2^j})$ and $R(\frac{k}{2^j})$ denote the quotient and remainder after division of k by 2^j, then the unavailable values of f at non integral indices can approximately be constructed as:

$$f(i+\frac{k}{2^j}) \cong \frac{1}{2^j}\begin{cases}(2^j-1)f(i+Q(\frac{k}{2^j}))+f(i+Q(\frac{k}{2^j})+1), & 0 < R(\frac{k}{2^j}) \le \frac{1}{2} \\ f(i+Q(\frac{k}{2^j}))+(2^j-1)f(i+Q(\frac{k}{2^j})+1), & \frac{1}{2} < R(\frac{k}{2^j}) < 1\end{cases}$$

Its advantage can be seen in Section(8.6).

1.9 Chapter9: A Special One-dimensional and Two-dimensional Wavelet Transform for Noise Reduction and Data Compression

Before processing signals and images effectively, we require the data be sufficiently noise free. A latest and efficient technique is the Wavelet Transform. In Section(9.1) it has been shown that "the repeated application of the average smoothing filter is equivalent to a wavelet transform". In Section(9.2) and Section(9.3) One-dimensional and Two-dimensional noise reducing wavelet transforms and their ANNs have been introduced along with coded programs to demonstrate the task of the ANNs. In Section(9.5) and Section(9.6), one-dimensional and two-dimensional data compressing wavelet transforms has been presented along with coded programs to test their performance.

1.9.1 A Wavelet Transform Deduced from Average Soothing Filter

An effective and traditional tool of smoothing data is the ***average filter***.

$$\bar{x}_i = \frac{x_i + x_{i+1}}{2} \quad \forall\ i = 1,2,\ldots,n-2\ ;$$

The processing of this filter can be speed up by applying it through a parallel processor but its repeated application leads to a feed back ANN. Avoiding feedback connections

lead us to deduce a wavelet transform from the mathematics of repeated application of the average smoothing filter. See Section(9.1).

1.9.2 Origin of Data Smoothing Wavelet Transform

Recall the definition of the new wavelet given in Equation(7.2.3) and the wavelets introduced in Section(6.16) which are given below:

(i) $\left\{\frac{1}{4},\frac{2}{4},\frac{1}{4}\right\}$; (ii) $\frac{1}{2}\left\{\frac{1}{8},\frac{2}{8},\frac{3}{8},\frac{4}{8},\frac{3}{8},\frac{2}{8},\frac{1}{8}\right\}$; (iii) $\frac{1}{4}\left\{\frac{1}{16},\frac{2}{16},\frac{3}{16},\frac{4}{16},\frac{5}{16},\frac{6}{16},\frac{7}{16},\frac{8}{16},\frac{7}{16},\frac{6}{16},\frac{5}{16},\frac{4}{16},\frac{3}{16},\frac{2}{16},\frac{1}{16}\right\}$

First application of the average filter is written as

$$y_i = \frac{x_i + x_{i+1}}{2} \quad \forall\ i = 1,2,\ldots,n-2\ ;$$

and the second one can be written as

$$z_i = \frac{y_i + y_{i+1}}{2} \quad \forall\ i = 1,2,\ldots,n-2\ ;$$

Combining the both, we have

$$z_i = \frac{\frac{x_i + x_{i+1}}{2} + \frac{x_{i+1} + x_{i+2}}{2}}{2} = \frac{x_i + 2x_{i+1} + x_{i+2}}{4} = (x_i, x_{i+1}, x_{i+2})(\frac{1}{4},\frac{2}{4},\frac{1}{4}),\ \ i = 1,2,\ldots,n-2$$

For the third and the fourth applications, we have

$$z_i = (x_i, x_{i+1}, x_{i+2}, x_{i+3})(\frac{1}{8},\frac{3}{8},\frac{3}{8},\frac{1}{8});\ \ z_i = (x_i, x_{i+1}, x_{i+2}, x_{i+3}, x_{i+4})[\frac{1}{2}(\frac{1}{8},\frac{4}{8},\frac{6}{8},\frac{4}{8},\frac{1}{8})],\ i = 1,2,\ldots,n-3;$$

respectively. A modified version is given by

$$z_i = (x_i, x_{i+1}, x_{i+2}, x_{i+3}, x_{i+4})[(\frac{1}{9},\frac{2}{9},\frac{3}{9},\frac{2}{9},\frac{1}{9})],\ \ i = 1,2,\ldots,n-3;$$

1.10 Chapter10: "Design Logic and Basic Structures of Parallel Phase Detecting Artificial Neural Networks "

We need to extract prominent features, peaks of cusps and bottoms of troughs along with their amplitudes and indices, for pattern recognition and for telecommunication of data in order to minimise the processing time, the communication cost, the size and the manufacturing cost of the relevant hardware. In this chapter, design topology and logic of parallel processing hardware are developed for these objectives. In Section(10.1). software checks of the procedures of data smoothing, detecting peaks and setting other data equals zero, reconstruction of data between peaks of signals are presented. In Section(10.2), a modification in the new zoom-out wavelet transform for reconstruction of

data is presented. A software check of developing Gaussian and half Gaussian in a new way is presented. In Section(10.3), a software check is presented for the procedure of setting image data other than peaks equals zero and reconstruction of the data. In Section(10.4), a procedure of finding the ordering structure of cusps and troughs is presented along-with a software check of finding the number of waves in a signal. In Section(10.6), a parallel processor for detecting peaks is designed and its working is explained by presenting an example. In Section(10.7-9), an optimality of the processor is addressed, its circuit is refined, and software check is presented. In Section(10.10), a circuit of the processor for data reduction by detecting troughs is developed and its software check is presented. In Section(10.11), a modified and finalised peak detecting parallel processor with software check is presented. In Section(10.12), an advanced dual circuit for peaks and bottoms detecting is designed. In Sections(10.13-15), three advanced ANNs; an advanced ANN classifier for the images/signals having fixed sizes, an advanced ANN classifier for telescopic images/signals, and an advanced ANN classifier based on the ordering structure of peaks and bottoms are designed and their topology and logic is explained.

1.11 Chapter11: Design Logic of Bit-wise Parallel Binary Adder Circuits

From the previous chapters it is clear that millions of similar processors are required almost in every ANN. Consider, for example, the two-dimensional data smoothing wavelet ANN shown in Figure(9.4.1), where the wavelet

$$\frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

has been used. There, each neuron of output layer performs first the direct multiplication of nine elements of the wavelet with the image elements which are under the window and then sums up the resulting products. Since there are millions of window placing, hence the millions of output neurons are required to perform millions of the weighted sums.

Using millions of the existing micro processors will not be a wise step. Because they are sequential, bigger, and costly. Some special purpose small parallel processors can be designed. This is the issue of this chapter.

Rejection of Old and Development of New Procedures for Artificial Neural Networks

The issue of this chapter is to reject the error minimisation methods of weights training and to develop a new theory of Artificial Neural Networks. It is proved in Section(2.1) that a neuron trained by an error minimisation method is deceive able. In Section(2.2), it is proved that the weights successfully produced by Windrow-Hoff delta rule are nothing more than the scaled versions of the components of samples. A new technique of weights construction is presented in Section(2.3). The weights enable a network for time incremental learning as well. In Section(2.4), a concept of straight line activation is presented. In Section(2.5), a concept of classes of objects and their threshold boundaries is presented. In Section(2.6), a procedure of dividing boundary of an image into two contours is presented. While in Section(2.7), a procedure of finding threshold boundaries a class is presented. The developments made in Section(2.3) through Section(2.7) are linked in Section(2.8). A general procedure is developed in order to have a straight line activated ANN classifier for classes of objects. To demonstrate the developments made in Section(2.8), a general example is presented in Section(2.9) through Section(2.11). The high force and the low force threshold activations for a class are presented in Section(2.9) and Section(2.10), respectively. Then both the high force and the low force activations are combined in Section(2.11) to classify the class. In Section(2.12), it is expressed that an image can be classified by finding contours from the continuation of peaks and bottoms of the image data. In Section(2.13), a procedure of finding limits of individual feature of a class of objects/events is developed. In Section(2.14) through Section(2.16), the concept, advantages/options, and a procedure of finding segment thresholds from the individual thresholds is presented. In Section(2.17-18), a procedure of finding weights and that of time incremental learning is presented. A binary coding transform with its inverse is presented in Section(2.19). This transform enables us to represent a signal with a single binary number.

2.1 A Drawback of Error Minimisation Methods of Weights Training

Graphical Representation: Each dotted line shown in Figure(2.1.1) represents a possible solution of the equation

$$\sum w_i x_i = TI .$$

In fact, there are infinite number of solutions of the equation; both for $w_i's$, and $x_i's$. A neuron being trained, by applying an error minimisation method like the delta rule, finds one of the solutions for $w_i's$. This fact leaves a serious drawback in the neuron. The trained neuron can easily get deceived by one of the infinite number of unseen signals that naturally exist.

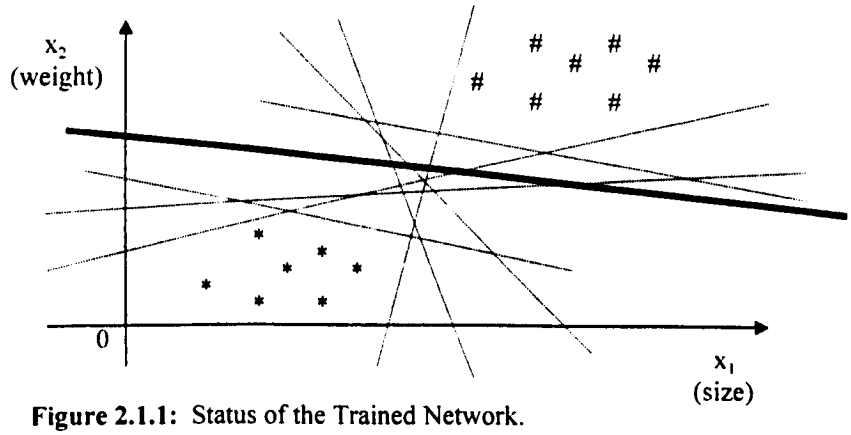


Figure 2.1.1: Status of the Trained Network.

Analytical Presentation: It will be proved in the following theorem that there exits a pattern (actually many) such that even a couple of its elements is sufficient to deceive the trained neuron.

Theorem: A necessary condition for the activation $f(\sum w_i x_i) = out$ of a neuron to be non deceive able is that each element x_i of a pattern $X = (x_0, x_1, ..., x_{n-1})$ belongs to a bounded region, where $w_i's$ are given.

Proof: Given that

$$f(\sum w_i x_i) = out \quad \Rightarrow \quad \sum w_i x_i = f^{-1}(out) = TI, \text{ say} \qquad (2.1.1)$$

By fixing all $x_i's$ except two say x_j and x_k, we have

$$w_j x_j + w_k x_k = T, \text{ where } T = TI - \sum_{\substack{i=0 \\ i \neq j \\ i \neq k}}^{n-1} w_i x_i \qquad (2.1.2)$$

$$\Rightarrow \qquad x_j = (T - w_k x_k)/w_j, \quad w_j \neq 0.$$

This means that $\forall\ x_k$, there corresponds an x_j such that Equation(2.1.2), and hence Equation(2.1.1) is satisfied. This implies that there exists no $M_j > 0$ such that

$$-M_j \leq x_j \leq M_j, \quad j = 0,1,\ldots,n-1.$$

This shows that there exist no two classes separable by their boundaries. Which is contradiction to reality. Hence the theorem.

Suggestions: This theorem implies that a neuron trained by an error minimisation method will not get deceived only if there is a boundary check in the ANN. Obviously this boundary separates a class from the others and it will therefore be called the decision boundary. There are two possibilities; (i) bounds check of individual data elements, (ii) bounds check by finding a finite number of pieces of straight lines enclosing the class. Moreover, the proper check can be done by reserving the first hidden layer for this purpose.

2.2 The Weights Produced by Delta Rule are Scaled Versions of Inputs

Let x_i^s and w_i denotes the *ith* component of the sample pattern X^s and its corresponding weight respectively, then the weights training by the delta rule is done as

Weights training: If *dout* and *aout* denote the desired and the actual output, respectively, then

$w_i(0) = x_i^s$
$aout = f(net)$
$\Delta = dout(0) - aout(0)$
$t = 1$
$while(\Delta != 0)\{$
    $w_i(t) = w_i(t-1) + \eta \Delta x_i^s$
    $aout = f(net)$
    $\Delta = dout(t) - aout(t)$
    $t = t + 1$
$\}$

Observe that

$$
\begin{aligned}
& t = 0 \\
& w_i(t) = x_i^s \\
& w_i^s(t+1) = w_i^s(t) + \eta \Delta x_i^s = (x_i^s + \eta \Delta x_i^s)(t) \\
& \qquad = (1 + \eta\Delta)(t) x_i^s \\
& \qquad = c_i^s(t) x_i^s, \quad c_i^s(t) \text{ are constants.}
\end{aligned}
$$

This shows that the weights of the trained network are scaled versions of the components of the sample pattern X^s.

2.3 A New Technique of Weights Construction

Let X, W, and out denote a pattern, a weight vector, and the target output respectively, of a neuron with the activation function f. Then for $w_i \in W$ and $x_i \in X$, the activation of the neuron is given by

$$
f\left(\sum_{i=0}^{n-1} w_i x_i\right) = out \quad \Rightarrow \quad \sum_{i=0}^{n-1} w_i x_i = f^{-1}(out) = T \tag{2.3.1}
$$

where T represents the target input of the neuron. This equation can be solved either by approximation methods or exact methods. We are interested in the later ones. There exists an infinite number of exact solutions. Three are given in the following.

Method 1: Consider the case when T is equally distributed over all non-zero elements of the pattern. If there are n non-zero elements, then we have

$$
w_i x_i = \frac{T}{n} \quad \Rightarrow \quad w_i = \frac{T}{n x_i}, \quad x_i \neq 0, \quad i = 1,....n \tag{2.3.2}
$$

This solution have no particular significance. Since the sufficiently small values of $x_i's$ result in local singularities. However, it can be applied to a class of patterns where each pattern X is such that $\forall \; x_i \in X$, $x_i \notin (-\varepsilon, \varepsilon)$, where ε is arbitrarily small positive number. A better solution is given below:

Method 2: Let us consider the successive two-dimensional perpendicular projections of the n-dimensional pattern X and allow the target input T to be distributed equally over all the non-zero projections. Mathematically, this means that

$$w_i x_i + w_{i+1} x_{i+1} = \frac{T}{m}, \quad i = 1,....n.$$

where w_i's are unknown weights, $x_i \in X$, and m is the number of non-zero projections. This equation can be solved for one of the w_i's after suitably fixing the other. We may like to solve for the weight that corresponds to greater of $|x_i|$ and $|x_{i+1}|$, especially when both are closer to zero. This minimises the occurrence of a local singularity. For example, if $|x_i| > |x_{i+1}|$ then we find w_i as follows

$$w_i = \frac{-w_{i+1} x_{i+1} + T/m}{x_i}, \quad i = 1,....n. \tag{2.3.3}$$

One can consider higher dimensional projections to deal with the patterns which have some features extremely close to zero. The higher the dimension of the projections, the higher the chance of non-occurrence of local singularities.

Method 3: The component x_i being divisors in Equations (2.3.2) and (2.3.3) may result in local singularities. In order to avoid such singularities and to achieve some other objectives from weights, we require the following:

(i) The weights should resemble with the pattern in some way. Mathematically, this means that W should be a function of X, say, $W = g(X)$.

(ii) The weights should be constructed such that they enable the trained network to time incremental learning, to complete the pattern when it come across a fractional part of the pattern, and to predict the future values from the existing ones, etc. To meet such properties, the function g must be bijective so that its inverse $X = g^{-1}(W)$ can be defined for the reconstruction of the pattern.

(iii) The functions g, and hence g^{-1}, should be defined such that there exits no chance of occurrence of any type of singularities in the construction of W and hence in the reconstruction or/and in the prediction of the pattern.

(iv) The function g should involve necessarily at least the three variables W, X, and T.

All these requirements can be achieved if we distribute T over the pattern proportionate with the energy spread of the pattern and construct the weights from this distribution. That is, for a given sample pattern X, the contribution of $x_i \in X$ to the energy of X equals x_i / λ, where $\lambda = \sum_{i=1}^{n} x_i^2$ and the spread of T proportionate with this contribution equals Tx_i/λ. Then, generally, we construct the weights $w_i = g(x_i, T, \lambda)$ as follows;

$$w_i = \frac{\alpha T x_i}{\lambda} + \beta, \quad i = 1,.....n \tag{2.3.4}$$

where α and β are constants and can be referred to as magnification and shift parameters respectively. The following example will show that Equations(2.3.4) provide a general criterion for weights construction and can be amended easily.

Example: Suppose that British Gas Company agreed to discourage the flow of population towards the well developed and over-crowded cities by charging higher rate per unit of gas such that the target income remains the same. Let x_i, w_i and r_i denote the estimated number of units of gas, rate per unit of gas and the rank of development plus crowd for the city i. The rate w_i (weight) relates to the rank r_i and the target income T, by

$$w_i \propto Tr_i \quad \Rightarrow \quad w_i = \frac{Tr_i}{\lambda}, \quad i = 1,.....n. \tag{2.3.5}$$

where λ is a constant of proportionality such that $\lambda = \sum_{i=1}^{n} x_i r_i$ and is defined to balance the amount of gas (energy) distributed over the whole population of the country. Multiplying by x_i and then summing over i, we have

$$\sum_{i=1}^{n} w_i x_i = T$$

This shows that if the rates (weights) computed from Equations(2.3.5) are used, the target income T will be the same. Although the above example is solved and the computed weights w_i's satisfy the conditions (i), (iii) and (iv). But in order to achieve property (ii), we have to consider that the company is also agreed (a) to control the wealth flow, (b) to encourage the energy saving and (c) to compensate the individual whose income is less.

In this case, let x_i denote the number of units consumed by a resident of the *ith* city of the rank r_i. Then the weights w_i's to be computed are related to T, r_i, and x_i, as given below:

$$w_i \propto Tr_i x_i \quad \Rightarrow \quad w_i = \frac{Tr_i x_i}{\lambda}, \quad i = 1,.....n. \tag{2.3.6}$$

$$\text{where } \lambda = \sum_{i=1}^{n} r_i x_i^2, \text{ and } \lambda \neq 0$$

Multiplying Equations(2.3.6) by x_i and then summing over i, we have

$$\sum_{i=1}^{n} w_i x_i = T$$

The weights computed from Equations(2.3.6) enables the network to reconstruct the pattern as follows:

$$x_i = \frac{\lambda w_i}{r_i T_i}, \quad r_i \neq 0, \quad T \neq 0$$

Obviously, all or any of $\lambda = 0$, $r_i = 0$, and $T = 0$ correspond to an aimless case. That is, there is no activity. Thus, from the above discussion, we set a general criterion for the best weights that

(i) the weights must be a constant multiple or/and a shifted version of the following entity;

$$\frac{Tr_i x_i}{\lambda}$$

(ii) more parameters, like r_i, can be involved depending on the nature of the problem for which a network is to be designed.

2.4 A Straight Line Activation

Let (x_1, x_2) be a two-dimensional pattern and (w_1, w_2) be the weights vector constructed from Equations(2.3.4) by setting $\alpha = T = 1$ and $\beta = 0$, then from the properties of a straight line we have

$$x_2 + mx_1 + c \begin{cases} > 0, & \text{if the point } (x_1, x_2) \text{ lies above the line.} \\ = 0, & \text{if the point } (x_1, x_2) \text{ lies on the line.} \\ < 0, & \text{if the point } (x_1, x_2) \text{ lies below the line.} \end{cases} \quad (2.4...$$

and

$$net = w_1 x_1 + w_2 x_2 + w_0 T = x_2 + mx_1 + c = 0$$

2.5 A Class of Objects and its Thresholding Boundary

A Class of Objects: A class of objects should be thought as a collection of microscopic and telescopic versions of a single object. For example, a set of lemons of all sizes but the same colours can be called a class of lemons of that colour. A class of n-dimensional patterns can be enclosed by a finite number of pieces of straight lines. For example, an irregular stone can be enclosed by a net of straight lines. The geometry of the net enclosing an object will provide the decision boundary of the object for classification. The larger the lengths of the pieces of straight lines, the less the number of nodes required for a classifier but more the chance of resemblance with other objects.

Digital Images: An image of a scene can have a number of different objects. Each object can be separated from its boundary. The boundary of an object is the union of pieces of straight lines joined end-to-end. For example, a digital image of a circle is the approximation achieved by joining end-to-end a finite number of small straight lines. Main features of such boundaries are the points of intersections the pieces of straight lines. As for as the high force and the low force thresholding is concerned, the boundary of an object can be divided into two parts; the upper boundary and the lower boundary.

Sufficient Number of Samples: Two sample patterns, the biggest and the smallest, are sufficient to have the decision boundary of a class. For example, a lemon can be classified by the boundary (check) of the class achieved from the biggest and the smallest lemon. It should be noted that in some cases it is not easy to decide the biggest and the smallest objects of a class. Some of the features of the objects may not be in well proportionate with that of the others. For example, the height by weight ratio varies from person to person. A proper procedure of computing the biggest and the smallest will be presented in Section(2.7).

Thresholding Boundary: The concept of the high force and the low force thresholding boundary will be clear in Section(2.6) and Section(2.7). However, the contour formed by joining end-to-end all the biggest (smallest) features of a class of sample patterns should be termed as the high (low) force thresholding boundary of the class.

2.6 Dividing Boundary of an Image into two Contours

Mathematically, a contour is a piece-wise continuous curve represented by a complex-valued function of a real variable. The outer boundary of an image should be divided (if required) into two contours (the high force and the low force contour) such that no two or more than two points of a contour can be crossed by a vertical straight line when drawn on the contour. In Figure(2.6.1) the straight lines $hf_1 - hf_5$ comprise the upper boundary (high force thresholding) and the straight lines $lf_1 - lf_5$ comprise the lower boundary (low force thresholding) of the aeroplane. The logic of Relations(2.4.1) is used to decide the lower and the upper boundaries.

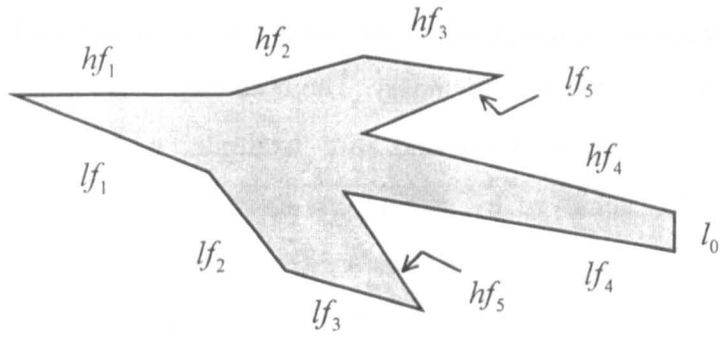


Figure 2.6.1: The high force and the low force boundaries.

For both the upper and the lower boundaries, we are interested in the angles of incidence, the lengths and the intersection points of the straight lines, but not interested in their positions. For example hf_5 and lf_5 has been placed in lengths and angles but not in positions with the upper and the lower boundaries respectively as shown in Figure(2.6.2). However, the order of the straight lines in working phase of the ANN must be the same as was in the training phase.

A vertical straight line like l_0 in Figure(2.6.2) can be ignored or otherwise can be included in the upper or lower boundary by twisting its one end either towards left or right.

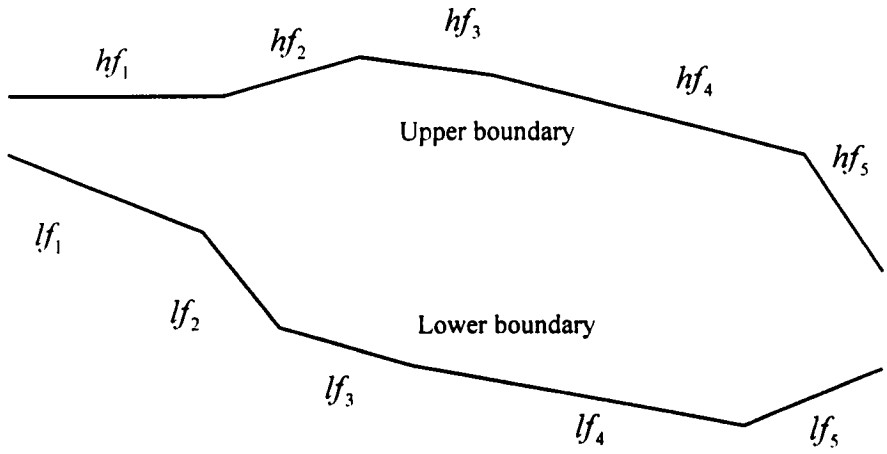


Figure 2.6.2: The lower and the upper boundaries of the aeroplane

2.7 Finding the threshold boundaries of a Class of Images

Although a scene (an image) may contain more than one objects, but for simplicity let the image consists of a single object. Let **A** be a class consisting of m images and let each image is symmetric about its central axis so that the number of pieces of straight lines enclosing the object from above be the same as that enclosing from below. Suppose this number is n. Then the high force and the low force threshold boundaries of the class are computed by following the three steps given below:

Step1: Collect all samples patterns of the class and construct the upper and the lower boundaries of each sample according to the procedure expressed in Section(2.6). See Figure(2.6.2) as an example. Let y_{ij} and z_{ij}, $i = 0,1,...,n$, $j = 1,2,...,m$

Step2: Let y_{ij} and z_{ij}, $(i = 0,1,...,n,\ j = 1,2,...,m)$ denote the *ith* corner point of the upper and the lower boundary of the *jth* sample respectively. Then develop two matrices $\mathbf{U} = [y_{ij}]$ and $\mathbf{L} = [z_{ij}]$.

Step3: Find the maximum of each column of **U** and the minimum of each column of **L** and then build two patterns, say $X\max$ and $X\min$, such that

$$X\max = (y_0, y_1, ..., y_{n-1}) \text{ and } X\min = (z_0, z_1, ..., z_{n-1})$$

where y_i and z_i denote the *ith* corner point of the high force and the low force thresholding boundary of the class, respectively, as shown in Figure(2.7.1).

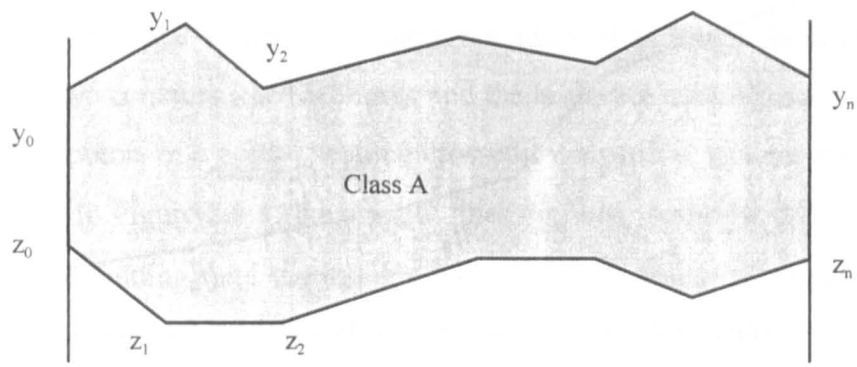


Figure 2.7.1: High and low force threshold boundaries of a class.

2.8 Linear Activation and Classification of Classes

Let us combine the following four developments:

(i) The idea, expressed in Method-2 in Section(2.3), of considering successive two-dimensional perpendicular projections of an n-dimensional pattern X,

(ii) Weights constructing Method-3 expressed in Section(2.3),

(iii) A straight line activation expressed in Section(2.4), and

(iv) The thresholding boundaries expressed in Section(2.7).

For a class of n-dimensional patterns X, let us considering the successive two-dimensional perpendicular projections of high force thresholding boundary, say $X\max$, and instead of finding weights by allowing the target input T to be distributed equally over all the non-zero projections $X\max$, we construct the weights by using Equation(2.3.4).

Let (w_i, w_{i+1}) be the weights for the *ith* projection of X : ax and T_i be the threshold value of the *ith* neuron. If we present the sample pattern (which is X max) to the network, then the weighted sum of each projection will produce its corresponding target input. That is.

$$w_i x_i^s + w_{i+1} x_{i+1}^s = T_i, \quad i = 1,.....n.$$

This implies that

$$net_i = w_i x_i^s + w_{i+1} x_{i+1}^s - T_i = x_{i+1}^s + m_i x_i^s + c_i = 0, \; i = 1,.....n.$$

Now let us present an actual pattern X^a to the network. If (x_i^a, x_{i+1}^a) is the *ith* projection of the actual pattern, then activations of the *ith* high force neuron and of the *ith* low force neuron of first hidden layer are given below:

High force *ith* neuron:

$$f(net_i) = \begin{cases} 1, \text{ if } (x_i^a, x_{i+1}^a) \text{ lies below or on the line } x_{i+1}^s + m_i x_i^s + c_i = 0. \\ 0, \text{ otherwise} \end{cases}$$

Low force *ith* neuron:

$$f(net_i) = \begin{cases} 1, \text{ if } (x_i^a, x_{i+1}^a) \text{ lies above or on the line } x_{i+1}^s + m_i x_i^s + c_i = 0. \\ 0, \text{ otherwise} \end{cases}$$

Equivalently and advantageously, we have

High force *ith* neuron:

$$f(net_i) = \begin{cases} 1, \text{ if } net_i \le 0; \text{ that is } (x_i^a, x_{i+1}^a) \text{ lies below or on the line } x_{i+1}^s + m_i x_i^s + c_i = 0. \\ 0, \text{ otherwise} \end{cases}$$

Low force *ith* neuron:

$$f(net_i) = \begin{cases} 1, \text{ if } net_i \ge 0; \text{ that is } (x_i^a, x_{i+1}^a) \text{ lies above or on the line } x_{i+1}^s + m_i x_i^s + c_i = 0. \\ 0, \text{ otherwise} \end{cases}$$

2.9 The High Force Thresholding

An example ANN acted upon high force threshold boundary is shown in Figure(2.9.2), while the threshold boundary is shown in Figure(2.9.1).

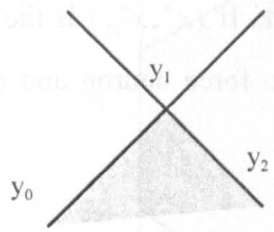


Figure 2.9.1 Upper boundary

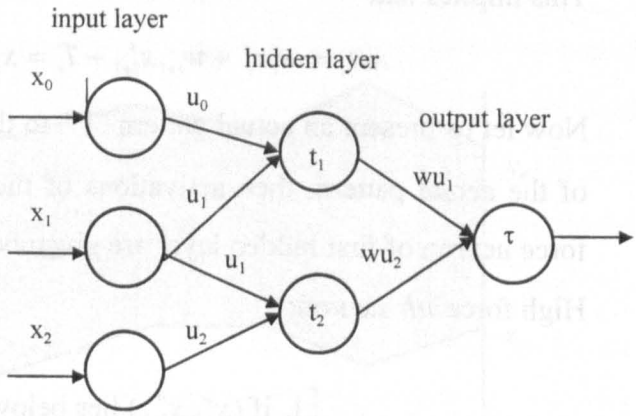


Figure 2.9.2 Circuit corresponding to upper boundary

Hidden layer:

- y_0 and u_0 are the bias input and the bias weight, respectively,
- If T is the given target input for the boundary pattern (y_0, y_1, y_2), then the weights denoted by u_i are given by

$$u_i = Ty_i / \sum y_i^2 \ , \ i = 1,2 \, .$$

- The threshold value, say t_j, for jth neuron is given by

$$u_{j-1}y_{j-1} + u_j y_j = (Ty_{j-1} + Ty_j) / \lambda = t_j, \ \ \lambda = \sum y_j^2 \ , \ \ j = 1,2 \, .$$

- $h(\alpha) = \alpha$ is the activation function for both hidden neurons. For a pattern $X = (x_0, x_1, x_2)$, the activation of the jth neuron is given by

$$netu_j = \sum u_j y_j \ ; \ \ h(netu_j) = \begin{cases} netu_j, & \text{if } netu_j \le t_j \\ 0, & otherwise \end{cases} \ , \ \ j = 1,2$$

Output neuron:

- Let τ denotes the threshold value for the output neuron, then the weights, denoted by wu_i are computed as

$$wu_i = \tau t_i / \sum t_i \,,\quad i = 1,2 \quad \text{so that} \quad \sum wu_i t_i = \tau \,.$$

- The delta function, denoted by f, is the activation function for the neuron. For the pattern $(netu_1, netu_2)$ coming from the hidden layer, activation of the neuron is

$$net = \sum wu_k netu_k \,;\quad f(net) = \begin{cases} 1, & when\ net \le \tau \\ 0, & otherwise \end{cases}$$

2.10 The Low Force Thresholding

An example ANN acted upon low force threshold boundary is shown in Figure(2.10.2), while the threshold boundary is shown in Figure(2.10.1).

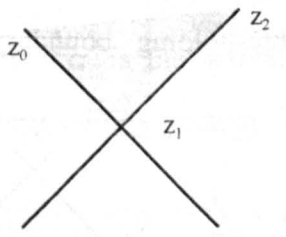


Figure 2.10.1 Lower boundary

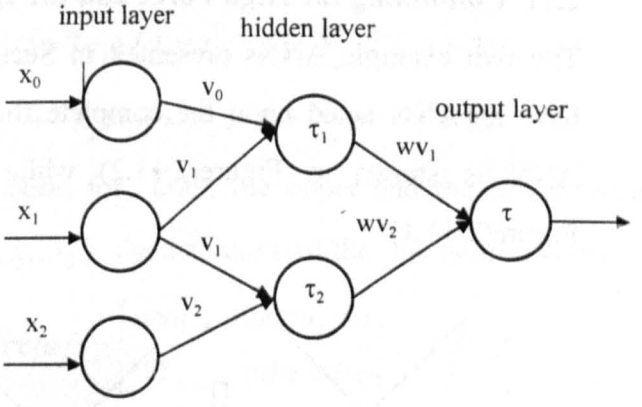


Figure 2.10.2 Circuit to implement lower boundary

Hidden layer:

- If z_0, v_0, and Γ are the bias input, the bias weight, and the given target input for the boundary pattern (z_0, z_1, z_2), then the weights denoted by v_i are given by

$$v_i = \Gamma z_i / \sum z_i^2 \,,\quad i = 1,2 \,.$$

- The threshold value τ_j, say, for the jth neuron is given by

$$v_{j-1} z_{j-1} + v_j z_j = (\Gamma z_{j-1} + \Gamma z_j) / \lambda = \tau_j \,,\quad \lambda = \sum z_i^2 \,,\quad j = 1,2 \,.$$

- $h(\alpha) = \alpha$ is the activation function for the both hidden neurons. For a pattern $X = (x_0, x_1, x_2)$, the activation of the jth neuron is given by

$$netl_j = \sum v_j z_j \,;\quad h(netl_j) = \begin{cases} netl_j, & \text{if } netl_j \ge \tau_j \\ 0, & \text{otherwise} \end{cases} \,,\quad j = 1,2 \,.$$

Output neuron:

- Let τ is threshold then the weights, denoted by wv_i are computed as

$$wv_i = \tau\tau_i / \sum \tau_i, \quad i = 1,2 \quad \text{so that} \quad \sum wv_i \tau_i = \tau .$$

- The delta function, denoted by f, is the activation function for the neuron. For the pattern $(netl_1, netl_2)$, coming from the hidden layer, activation of the neuron is given by

$$net = \sum wv_i netl_i, \quad i = 1,2 \quad \text{and} \quad f(net) = \begin{cases} 1, & \text{if } net \geq \tau \\ 0, & \text{otherwise} \end{cases}$$

2.11 Combining the High Force and the Low Force Thresholding

The two example ANNs presented in Section(2.9) and Section(2.10) are combined to have an ANN acted upon the complete threshold boundary of a class of objects. The ANN is shown in Figure(2.11.2) while the thresholding boundary is shown in Figure(2.11.1).

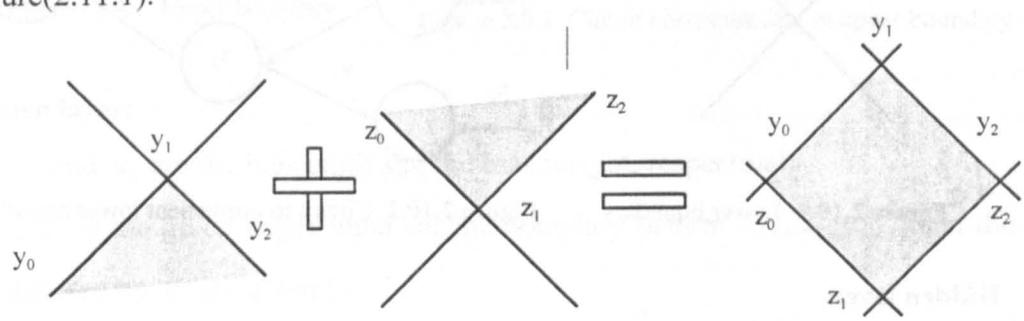


Figure 2.11.1 Full boundary of an object enclosed by four straight

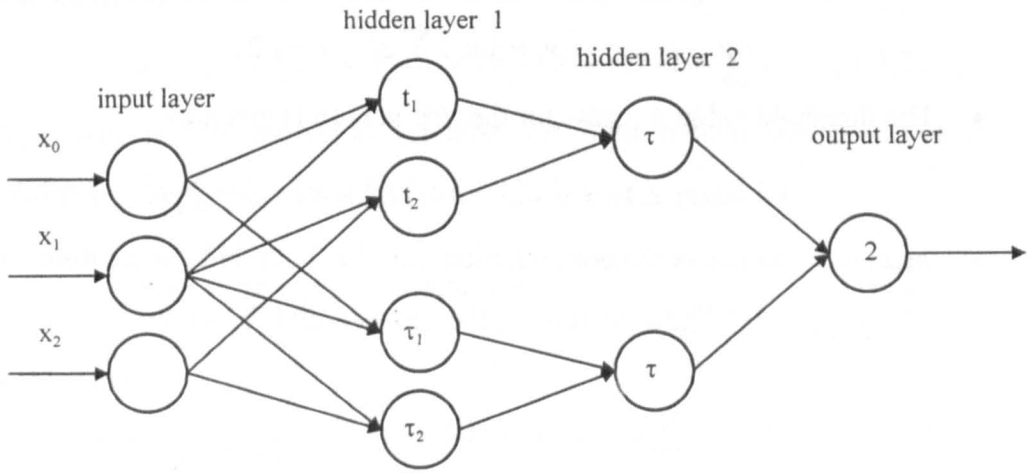


Figure 2.11.2 Model of an ANN classifier for one object enclosed four straight lines

First Hidden layer:

- y_0, z_0 and u_0, v_0 are the bias inputs and the bias weights, respectively.
- If T and Γ are the given target inputs for the upper and the lower boundary patterns (y_0, y_1, y_2) and (z_0, z_1, z_2), respectively, then the weights denoted by u_i and v_i are computed by using equation (2.3.4) and are given by

$$u_i = Ty_i / \sum y_i^2 , \quad i = 1,2 .$$

$$v_i = \Gamma z_i / \sum z_i^2 , \quad i = 1,2 .$$

- The threshold values t_j and τ_j, say, for the jth neurons corresponding to the upper and the lower boundary neurons , respectively, are given by

$$u_{j-1}y_{j-1} + u_j y_j = (Ty_{j-1} + Ty_j) / \lambda = t_j , \quad \lambda = \sum y_j^2 , \quad j = 1,2 .$$

$$v_{j-1}z_{j-1} + v_j z_j = (\Gamma z_{j-1} + \Gamma z_j) / \lambda = \tau_j , \quad \lambda = \sum z_i^2 , \quad j = 1,2 .$$

- $h(\alpha) = \alpha$ is the activation function for both, the upper and the lower, boundary neurons. For a pattern $X = (x_0, x_1, x_2)$, the activation of the jth neuron is given by

$$netu_j = \sum u_j y_j ; \quad h(netu_j) = \begin{cases} netu_j, & \text{if } netu_j \le t_j \\ 0, & otherwise \end{cases} , \quad j = 1,2 .$$

$$netl_j = \sum v_j z_j ; \quad h(netl_j) = \begin{cases} netl_j, & \text{if } netl_j \ge \tau_j \\ 0, & \text{otherwise} \end{cases} , \quad j = 1,2 .$$

Second hidden layer

- Let τ denote is value for both neurons in the layer, then the weights, denoted by wu_i and wv_i , for the upper and the lower boundary , respectively, are computed as

$$wu_i = \tau t_i / \sum t_i , \quad i = 1,2$$

$$wv_i = \tau\tau_i / \sum \tau_i , \quad i = 1,2 .$$

Obviously, $\sum wu_i t_i = \tau$ and $\sum wv_i \tau_i = \tau$.

- The delta function, denoted by f, is the activation function for both the neurons of the layer. For the inputs $(netu_1, netu_2)$ and $(netl_1, netl_2)$ coming from the first hidden layer, the activation of the neurons are given by

$$net = \sum wu_k netu_k \; ; \; f(net) = \begin{cases} 1, & \text{if } net \le \tau \\ 0, & otherwise \end{cases}, \quad i = 1,2$$

$$net = \sum wv_i netl_i \; ; \quad f(net) = \begin{cases} 1, & \text{if } net \ge \tau \\ 0, & \text{otherwise} \end{cases}, \quad i = 1,2$$

Output layer:

Each weight equals 1; threshold value equals 2; and activation is given by

$$f(net) = \begin{cases} 1, & \text{if } net = 2 \\ 0, & \text{otherwise} \end{cases}$$

where *net* denote the net input of the neuron.

2.12 Image Data and Extracting Contours

A two-dimensional digital image can be considered as $n \times n$ matrix of grey levels and graph of data looks like a hilly area. There may be some other ways for image classification but analysing for peaks of hills and bottoms of valleys seems a better one. It has been observed practically that both the peaks and the bottoms of the rows have some meaningful continuation from row to row. See for example, the output data by executing the software program **tfpidz.cpp** listed in Appendix(5.6). This continuation forms contours. Mathematically, a contour is a piece-wise continuous curve represented by a complex-valued function of a real variable. These contours should be divided (if required) into sections (the high force and the low force thresholding boundaries) such that no two or more than two points be crossed by a vertical straight line when drawn on a section. There are two advantages: (i) the sections can further be processed for data reduction by picking their corner points and discarding the data between the corners, (ii) picking their relative positions will result in getting ride of the difference of physical positions of samples and the actual images at scenes. The procedure to find the peaks and bottoms and other relevant developments will be presented in Chapter(10). However, the concept of dividing the entire boundary into sections of contours is expressed in Section(2.6).

2.13 Finding Limits of Individual Features of a Class of Objects/Events

Let **A** be a class consisting of n (m) objects/events and each object/event is recognised by 1 feature, say x_j. Since the feature is variant and the variation is bounded, therefore, for jth object, there exist two real numbers lub_j and glb_j such that $\text{glb}_j \le x_j \le \text{lub}_j$. Objective of this section is to find glb_j and lub_j. Then the high force and the low force threshold limits of the class are computed by following the three steps given below:

Step1: Collect m (all) samples of each object. That is, the set $(x_{1j}, x_{2j}, \ldots, x_{mj})$ of m samples of jth object.

Step2: Build an m x n matrix $\mathbf{A} = [x_{ij}]$, ($i = 1,2,\ldots,m$, $j = 1,2,\ldots,n$) with the samples on its columns.

Step3: Find the maximum and the minimum of each column of **A** and then build two patterns, say $X\max$ and $X\min$, such that

$$X\max = (y_1, y_2, \ldots, y_n) \text{ and } X\min = (z_1, z_2, \ldots, z_n)$$

where y_i and z_i denote the ith high force and the low force threshold limit, respectively, of the class, as shown in Figure(2.7.1).

An ANN acted upon this type of limitation is shown in Figure(4.7).

Generalisation: Above procedure can repeatedly be applied for each feature when each object of the class is recognised by more than one features.

2.14 From Individual Thresholds to Segment Thresholds

Individual Limits: Most of the physical objects are recognised through their features, like weight, volume, moss, and colour etc. For a particular class of objects, each feature of an object has its individual upper and lower limits. The union of the individual limits of all the features of an object separates the object from the others. Hence we need a layer of neurons consisting of the high force and the low force thresholding to check the limitations of a class of objects. For example, see the first hidden layer of the ANN shown in Figure(4.7). A method of finding such limitations is presented in Section(2.13).

Segment Thresholds: Objects and events may have a large number of components/ features. Activating for each component will require a separate hardware circuitry. In order to minimise design topology of ANNs, the pattern of features can be divided into suitable segments. There are a number options for activation of the segments. This issue will be discussed in Section(2.15). A method of finding segment thresholds is given in Section(2.16). However, for a natural example see chapter4 where instead of activating the individual muscle fibres, a group is activated by a single branch of the feeding axon.

2.15 Ignoring Bound Checks of Objects in a Hull Set

Let us suppose that there are some classes of objects such that the activation (outputs) of all the classes lie in an acceptable pre-decided range, say $[a,b]$. Then such classes can be grouped into a composite class, usually called a Hull set. Irrespective of the fact that the corresponding objects of all the classes have some relationship between their activation (or limits of features) or not. A single unit of an ANN can be developed to classify an object belonging to any of the classes. There are at least four options:

Option-1: The best option for accurate classification is to deal with each member class separately. For example, each muscle is dealt as a separate class in neuromuscular ANN classifier shown in Figure(4.7).

Option-2: When there is some relationship between the corresponding objects of the classes. In this case, we can merge all the classes into one class so that the procedure of finding threshold limit expressed in Section(2.13) be applicable.

Option-3: Instead of applying bound checks for each individual feature of each object of a member class, we can apply a bound check on the weighted sum of all the features of an object. But it should be remember that the network may be deceived in the sense expressed in Section(2.1).

Option-4: We can, further, ignore the bound checks on the weighted sums as expressed in Optio-3 above. And apply bound checks only on the weighted outputs of the member classes of the Hull set. Again, it should be noted that the chance for the network to be deceived is increased as compared to that of Option-3 given above. But for some special

applications, where some particular objects are allowed to be input, this option can be used. Its purpose is to minimise design topology of the network.

Note: Visualise that the design topology of networks acted upon Option-3 and Option-4 will resemble partially with those trainable by error minimisation methods.

2.16 Finding Segment Thresholds from Individual Thresholds

The concept of sampling a continuous function at discrete values of its independent variable can be used in activating upon the segments. Consider the high force threshold limit signal $X\max = (y_1, y_2, ..., y_n)$ and let t denotes its index variable. If we partition the discrete interval $\mathrm{I}: 1 \le t \le n$ into, say, m subintervals $\mathrm{I}_k : t_{k-1} \le t \le t_k$ such that $2 \le (t_k - t_{k-1}) \le n$, $k = 1,2,...m$, $m \le n$ then the number of neurons of the layer reduces to the number of subintervals. But it does not mean that the validity of the components having indices other than t_k may not be checked. Activation of *kth* neuron will be a particular straight line and will stand as common activation for all the components whose indices lie in the interval I_k. An acceptable difference of the values of $y_t's$, denoted by v is helpful to find the lengths, denoted by l_k, of the subintervals $\mathrm{I}_k : t_{k-1} \le t \le t_k$. Then the *kth* straight line of length equals l_k is drawn through the greatest most two components of the segment. An example ANN is shown in Figure(3.7b). This procedure is also applicable for low force threshold patterns with the only difference the *kth* straight line of length equals l_k is drawn through the smallest most two components of the segment. For both, the high force and the low force thresholds, there is no harm if each straight line be drawn parallel to time-axis (when v is small) and passing through the greatest component of the segment as shown in Figure(2.16.1)..

Software 2.16.1: A software program named **tfsegs.cpp** (test finding of segments) is developed to find segment thresholds. It is listed in appendix(3.6) and its outputs are shown in Figure(2.16.1).

Figure 2.16.1: Outputs of tfsegs.cpp; the input cosine signal and the linear segmented cosine.

It should be noted that the value of v is a positive real number and is related to the amplitudes and frequency of inputs $y_i's$ under the following approximation:

For a cosine signal having frequency 128/4 (that is, (length of signal)/(number of waves)), we have the following relation

$$v \le \frac{\max\{|y_i|, t = 1,2,.....n\}}{2} - \frac{err}{6}, \text{ where } err \text{ is a constant to be determined.}$$

Similarly, the lower limit of bound v can be found.

2.17 Finding Weights for Thresholds Patterns

The threshold patterns $X\max$ and $X\min$, computed in Section(2.7), Section(2.13) and Section(2.16) (or computed in some other way), require weights for their activation. Simply, apply Equations(2.3.4). For example, by setting $\alpha = T = 1$ and $\beta = 0$, we have

$$\text{Weights for } X\max : \; wu_i = \frac{y_i}{\lambda}, \; i = 1,....n, \text{ where } \lambda = \sum_{i=1}^{n} y_i^2 .$$

$$\text{Weights for } X\min : \; wv_i = \frac{z_i}{\lambda}, \; i = 1,....n, \text{ where } \lambda = \sum_{i=1}^{n} z_i^2 .$$

2.18 Procedure of Time Incremental Learning

The network trained by using an error minimisation method of weights training do not have the ability of time incremental learning. On the other hand, the ANNs acted upon threshold limits, threshold boundaries, segment thresholds, etc. have the ability of time incremental learning. That is, the ability to learn for new patterns without requiring the previously known patterns.

Step1: Collect all new samples and compute $Y\max$ and $Y\min$ by using the same method that was used earlier to compute $X\max$ and $X\min$ given in Section(2.7), Section(2.13), and Section(2.16).

Step2: Upgrade the patterns $X\max$ and $X\min$ by comparing with $Y\max$ and $X\min$, respectively, as follows;

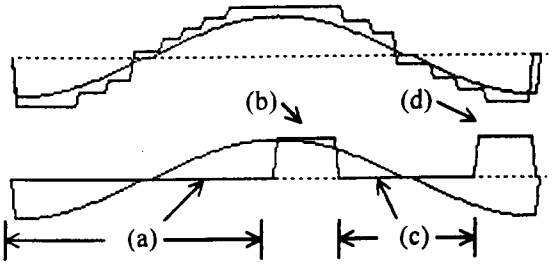
```
for(i = 1;i <= n;i++){
    if(Ymin[i] < Xmin[i])Xmin[i] = Ymin[i];
    if(Ymax[i] > Xmax[i])Xmax[i] = Ymax[i];
}
```

2.19 Developing a Binary Transform and its Inverse

In this section a binary coding transform (BT) and its inverse is presented. A signal is particularly divided into segments and each segment can be shifted with its center at zero axis. Length and shift values of each signal are preserved for reconstruction. The segmented and shifted signal is passed to the BT transform in order to have a single three stage binary number representing the complete original signal. The length of each segment depends upon a pre-decided constant , say v, which is the acceptable difference of values of components of the signal to be transformed. The smaller the value of v, the greater the accuracy and better the reconstruction. It should be noted that the value of v is positive real number and is constrained by the amplitudes and frequency of inputs $y_i's$ under the relation expressed in Section(2.16).

Software 2.19: A software program named bt.cpp (binary transform) has been developed to test and demonstrate the development of binary transform. The program is listed in Appendix(3.7) and its outputs are shown in Figure(2.19.1). It should be noted that the code for reconstruction is not included in the program due to some uncertain error in the output; see Figure(2.9.1B). However, it can be computed easily by one of the following two alternative methods:

- by adding the shift (that is, $\max y$ in the program) values in the corresponding transformed segments and then smoothing.
- by adding the shift values only in the initial and the terminal points of the corresponding transformed segments and then reconstructing other data elements by using the new wavelet transform (thgistic.cpp) presented in section Software(10.2.2) .



(A) The input cosine Wave and its segment thresholds.

(B)The input cosine wave and its Binary Transform. (partially abnormal output: sections (a) and (d) are abnormal while (b) and © can be thought of normal.

Figure 2.19.1: Outputs of bt.cpp to demonstrate the development of Binary Transform.

Rejection of Old and Development of New Classifier Artificial Neural Networks

It is shown in Section (3.1) that there exists no exact method of weights construction like that developed in Section(2.3) and that weights training is done mostly by using iterative trial and error methods. In Section(3.2), a series of new ANNs are presented. Although, the ANNs are better than the Hopfield network, yet they can not be considered better than the other ANNs newly presented in a number of chapters of this thesis. Their purpose of development is to prove that the weights of the Hopfield network are either destructive or redundant and that the Hopfield network does not qualify as a reasonable network. In Section(3.3), it is shown that the Boltzmann Machine, like the Hopfield network, does not qualify as a reasonable network. Moreover, it is shown that the weights of the trained Boltzmann Machine are shifted version of the weights of the Hopfield network. In Section(3.4), a new multiclass linear classifier ANN is developed. The purpose of the ANN is to present a new design topology for classifier ANNs. The ANN also demonstrate the application of the new weights constructing formula given in Equations(2.3.4) and that the linear activation given in Section(2.4). In Section(3.5), A multiclass ANN acted upon segment thresholds is developed. The ANN demonstrates the advantage of reducing the design topology of classifier ANNs by using a straight line activation and that the high force and the low force segment threshold decision boundaries. In Section(3.6), A series of multiclass XOR ANNs are presented. The ANNs are straight line activated and have two options of design topology; and hence of activation. In Section(3.7), a training procedure of the XOR ANN for n objects is presented. In Section(3.8), A new XOR ANN as a detector for m objects. Its practical use is to detect the happening of anyone of the m events in the industrial automation, presence of anyone of the m known persons or objects, etc. In Section(3.9), A Generalised XOR ANN detector for m objects is presented. Its design topology demonstrates the situations when number of features of different objects or events are not the same.

3.1 The Error Minimisation Training Methods can no longer exist with ANNs!

This will be shown that the author of [1], like the others and unlike us, could not find the way out of achieving weights other than using the error minimisation rules. The author expressed the failure of the perceptron (an artificial neuron) in Section(3.7) of his book as:

> "The failure of the perceptron to successfully solve apparently simple problems such as the XOR one was first demonstrated by Minsky and Papert in their book Perceptrons."

Then the author expressed a way out in Section(4.2.1) as:

> "How are we to overcome the problem of being unable to solve linearly inseparable problems with our perceptron? An initial approach would be to use more than one perceptron, each set up to identify small, linearly separable section of the inputs, then combining their outputs into another perceptron, which produce a final indication of the class to input belongs."

Further the author expressed that

> "This seems fine on first examination, but a moment's thought will show that this arrangement of perceptrons in layers will be unable to learn."

Although, the author stated next the reasons of this conclusion, but the basic mistake may have occurred in Section(2.7) of his book. Where the author expressed "This is clearly a *straight line* decision boundary...." but he could not realised the position of this boundary as he stated:

> "We can see that the function does indeed give us a decision boundary but we are no closer to realise the position of this boundary or finding the correct components for the weight vector."

Then the author stated his conclusion in the concrete form as:

> "Thus far we have proved that if we have the correct value for the weight vector we can indeed perform the discriminating process and set the position of the decision boundary."
>
> "What we have not shown yet is the critical part - namely finding the weight vector. This, unfortunately, is not a trivial problem! It is most usually found by iterative trial and error methods that modify the weight values according to some error function."

Author's conclusion verifies that there exists no exact method of constructing weights and that the mostly used methods are iterative trial and error methods. It has been proved in Section(2.1) that a neuron trained by an error minimisation method can be deceived. If some design topology of a network guarantees that a neuron can not be deceived, even then the methods can not be justified. Because, it is proved in Section(2.2) that the weights successfully produced by the Windrow-Hoff delta rule are nothing more than the scaled versions of the components of the samples. A meaningful scaling can appropriately be done by exact calculation. For example, the new weights constructing formula given in Equations(2.3.4) provides weights which are the normalised (by energy) versions of samples.

3.2 The Hopfield Network does not Qualify as a Reasonable Network

It will be shown that the Hopfield network does not qualify as a reasonable network through a series of results in the following subsections.

3.2.1 An Alternate A1 of the Hopfield Network

An alternate of the Hopfield network is presented with a difference of one connection and all of the weights. The models of both the networks are shown in Figure(3.2.1) and Figure(3.2.2) and their iterative models are shown in Figure(3.2.3) and Figure(3.2.4).

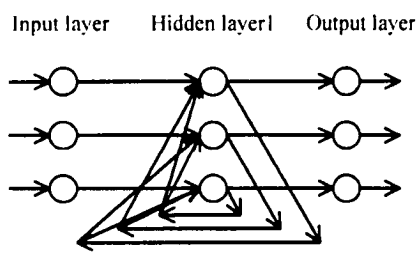


Figure 3.2.1: The Hopfield Network.

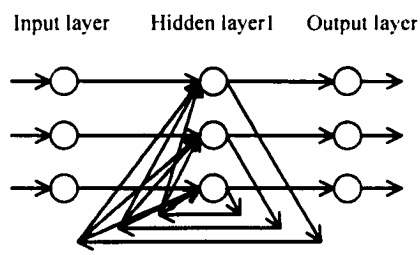


Figure 3.2.2: A1 Network.

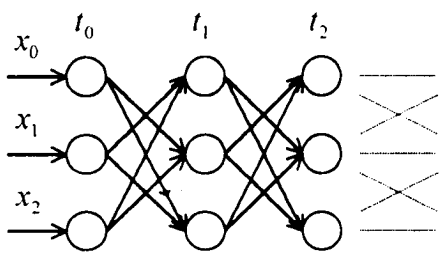


Figure 3.2.3: The Hopfield Iteration Model.

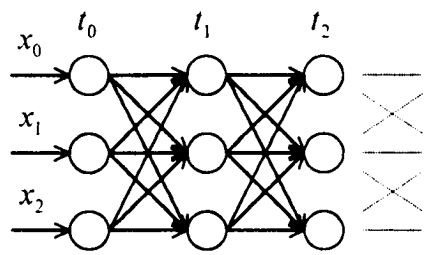


Figure 3.2.4: A1 Iteration Model.

The weights construction and the activation of A1 network is given below:

Weights: Construct weights w_{ij} from samples X^j by using Equations(2.3.4) as

$$w_{ij} = \frac{x_{ij}}{\lambda_j};\ \ \lambda_j = \sum_{i=0}^{n-1} x_{ij}^2;\ \ i = 0,1,....n-1;\ \ j = 1,2,....m$$

where x_{ij} denotes the *ith* component of *jth* sample.

Input Layer: Present an actual pattern on connections of the input layer. That is,

$$x_i(0) = x_i,\ \ i = 0,1,....n-1$$

Hidden layer1: Activate and iterate, if necessary, as

while($a_j < 0$){

$$anet_j = \sum_{i=0}^{n-1} w_{ij} x_i;\ \ a_j = anet_j - 1;$$

$$x_i(t+1) = f(a_j) = \begin{cases} -1, & \text{when } a_j < 0 \\ x_i(t), & \text{when } a_j \geq 0 \end{cases},\ \ j = 1,2,....m$$

}

Output Layer: Output the values received from the hidden layer, that is, output x_i's.

The A1 network is, of course, not the Hopfield network but performs the task of his network.

3.2.2 Another Alternate A2 Network of the Hopfield Network

Another alternate of the Hopfield network is named A2. Its design topology is the same as that of A1 which is shown in Figures(3.2.2). The details of this network will be used subsequently in showing the redundancy of weights of the Hopfield network.

Let net_i^h denote the weighted sum of the *ith* neuron of the Hopfield network in a general sense and let net_i^{sh} stand for net_i^h when the Hopfield weights are the simplest as given in Equation(3.2.1). Moreover, $snet_i^h$ and $anet_i^h$ can be referred as the weighted sum associated with a sample and with an actual pattern, respectively, for the Hopfield network. Then for a single n-dimensional sample pattern X^1, we have

Weights: The Hopfield weights w_{ij}, $i \neq j$ and one different weight $w_{ii} = 1$, are given as

$$w_{ij} = \begin{cases} x_i^1 x_j^1, & when\ i \neq j \\ 1, & when\ i = j \end{cases}, \quad i = 0.1,.....n-1, \quad j = 0.1,.....n-1. \tag{3.2.1}$$

Input Layer: Present an actual pattern on connections of the input layer. That is,

$$x_i(0) = x_i, \quad i = 0.1,.....n-1.$$

Before expressing activation of the first hidden layer, observe that when the sample is passed to the trained network, then

$$snet_i = \sum_{j=0}^{n-1} w_{ij} x_j^s = 1 \quad \text{and} \quad a_i = snet_i - 1 = 0,$$

For the Hopfield network ($w_{ii} = 0$), we have

$$snet_i^{sh} = \sum_{\substack{j=0 \\ j \neq i}}^{n-1} w_{ij} x_j^s .$$

So that

$$a_i = snet_i^{sh} - 1 + x_i^s = 0, \quad i = 0,1,....n-1.$$

Hidden layer1: Activate and iterate, if necessary, as

while(a_i ! = 0){

$$anet_i^{sh} = \sum_{\substack{j=0 \\ j \neq i}}^{n-1} w_{ij} x_j \ ; \quad a_i = anet_i^{sh} - 1 + x_i \tag{3.2.2}$$

$$x_i(t+1) = f(a_i) = \begin{cases} -1, & \text{when } a_i \neq 0 \\ x_i(t), & \text{when } a_i = 0 \end{cases}, \quad i = 0,1,2 .$$

}

Output Layer: Output the values sent by the hidden layer, that is x_i, $i = 0,1,....n-1$.

3.2.3 Another Alternate A3 of the Hopfield Network

Another alternate of the Hopfield network is named A3. Its design topology is the same as that of A1 and A2 which is shown in Figures(3.2.2). The details of this network will be used, in addition to that of A2, subsequently. For a single n-dimensional sample pattern X^1, we have

Case-1: When there is only one pattern to be classified and weights be achieved from the shifted version of the sample. Let β denotes a shift value for the sample X^1. The actual

definition and the purpose of β will be clear in Section(3.2.4), however, weights can be defined as

$$w'_{ij} = \begin{cases} x^1_i(x^1_j + \beta / x^1_j), & when\ i \neq j \text{ and } x^1_j \neq 0 \\ x^1_i x^1_j, & when\ i \neq j \text{ and } x^1_j = 0,\ i = 0,1,\ldots,n-,\ j = 0,1,\ldots,n- \\ 1, & when\ i = j \end{cases}$$

Note that when $\beta = 0$ then $w'_{ij} = w_{ij}$, where w_{ij}'s are the same as defined in Relations(3.2.1).

$$anet'^{h}_i = \sum_{\substack{j=0 \\ j\neq 0}}^{n-1} w'_{ij} x_j = \left(\sum_{\substack{j=0 \\ j\neq 0}}^{n-1} w_{ij} x_j \right) + (n-1)x_i\beta = anet^{sh}_i + (n-1)x_i\beta$$

$$\begin{aligned} a_i &= anet^{sh}_i - 1 + x_i + (n-1)x_i\beta \\ &= anet^{sh}_i - 1 + x_i[1 + (n-1)\beta] \\ &= anet^{sh}_i - 1 + cx_i,\ c = 1 + (n-1)\beta,\ i = 0,1,\ldots,n-1. \end{aligned}$$

In order to get closer to the Hopfield network we consider patterns having components equals 1 and -1. Since the constant c appeared as a multiple of x_i, therefore if $c \neq 1$ then its existence is destructive and if $c = 1$ then the cause $(\beta \neq 0)$ of its appearance is a redundancy. To remove the cause, we set $c = 1 + (n-1)\beta = 1 \Rightarrow \beta = 0$ (for non-trivial case $n > 1$). This results in reducing this case to A2-network expressed in Section(3.2.2).

Case-2: When there is only one pattern to be classified and each weight be achieved from the shifted version of the corresponding component of the sample, unlike a common shift for all the components in case-1. Let β_i denotes the shift value of the *ith* component of the sample, then weights can be defined as

$$w'_{ij} = \begin{cases} x^1_i(x^1_j + \beta_i / x^1_j), & when\ i \neq j \text{ and } x^1_j \neq 0 \\ x^1_i x^1_j, & when\ i \neq j \text{ and } x^1_j = 0,\ i = 0,1,\ldots,n-,\ j = 0,1,\ldots,n-1 \\ 1, & when\ i = j \end{cases} \quad (3.2.3)$$

Defining weights in this way enables us to hide the presence of samples other than X^1 which are used in generating weights for the Hopfield network. How ?, this will be clear in Section(3.2.4).

$$anet_i^{th} = \sum_{\substack{j=0 \\ j\neq 0}}^{n-1} w_{ij}^{'} x_j = \left(\sum_{\substack{j=0 \\ j\neq 0}}^{n-1} w_{ij}^{'} x_j \right) + (n-1)x_i\beta_i = anet_i^{sh} + (n-1)x_i\beta_i \qquad (3.2.4)$$

$$\begin{aligned} a_i &= anet_i^{sh} - 1 + x_i + (n-1)x_i\beta_i \\ &= anet_i^{sh} - 1 + x_i[1 + (n-1)\beta_i] \\ &= anet_i^{sh} - 1 + c_i x_i, \quad c_i = 1 + (n-1)\beta_i, \quad i = 0,1,\ldots,n-1. \end{aligned} \qquad (3.2.5)$$

Again, the constant c_i appeared as a multiple of x_i, and as it is explained in case-1, if $c_i \neq 1$ then its existence is destructive and if $c_i = 1$ then the cause $(\beta_i \neq 0)$ of its appearance is a redundancy. If we set $c_i = 1 + (n-1)\beta_i = 1$ then $\beta_i = 0$ (for non-trivial case $n > 1$) results in reducing this case, as well, to A2-network expressed Section(3.2.2).

3.2.4 Redundancy of Weights of the Hopfield Network

The weights of the Hopfield network can be visualised in a matrix form. A matrix $\mathbf{A}^s$ having $n \times n$ dimension can be developed for the sample pattern X^s by writing the sample, after multiplying it with its jth component, on the jth column and setting the main diagonal elements equals zero. Then the weights matrix, denoted by $\mathbf{W}$, corresponding to all sample patterns is given as

$$\mathbf{W} = \sum_{s=1}^{m} \mathbf{A}^s = \left[\sum_{s=1}^{m} x_i^s x_j^s \right] = \left[w_{ij}^{'} \right], \quad i = 0,1,\ldots,n-1; \quad j = 0,1,\ldots..n$$

where $w_{ij}^{'}$, weights of the Hopfield network, are given explicitly as

$$w_{ij}^{'} = \begin{cases} \sum_{s=1}^{m} x_i^s x_j^s, & i \neq j \\ 0, & i = j \end{cases}; \; i = 0,1,\ldots,n-1; \quad j = 0,1,\ldots..,n. \qquad (3.2.6)$$

If β_i used in Relations(3.2.3) be defined as

$$\beta_i = x_j^k \sum_{\substack{s=1 \\ s\neq k}}^{m} x_i^s x_j^s; \quad i \neq j; \quad i = 0,1,\ldots,n-1; \quad j = 0,1,\ldots,n \qquad (3.2.7)$$

then Relation(3.2.3) can be written for some $k = 1.2,\ldots,m$ in reduced form as

$$w_{ij}^{'} = \begin{cases} x_i^k (x_j^k + \beta_i / x_j^k), & i \neq j \\ 1, & i = j \end{cases}, \quad i = 0,1,\ldots,n-1; \quad j = 0,1,\ldots,n \qquad (3.2.8)$$

Two things are important here: (i) the definition of β_i's given in Relations(3.2.7) enables the network A3 fit for m patterns which was considered there for one pattern, (ii) the only difference of the weights given in Relations(3.2.6) from those given in Relations(3.2.8) is

$$w_{ii}^{'} = \begin{cases} 1, & \text{for A3 network} \\ 0, & \text{for Hopfield network} \end{cases}, \quad i = 0,1,\ldots,n-1.$$

Apart from any comparison, the values $anet_i^h$ and a_i^h of the Hopfield network should be expressed as given in Relations(3.2.4) and Relations(3.2.5). That is,

$$anet_i^h = \sum_{\substack{j=0 \\ j\neq i}}^{n-1} w_{ij}^{'} x_j = \left(\sum_{\substack{j=0 \\ j\neq i}}^{n-1} w_{ij} x_j \right) + (n-1)x_i\beta_i = anet_i^h + (n-1)x_i\beta_i \qquad (3.2.9)$$

$$a_i^h = anet_i^h - 1 + (n-1)x_i\beta_i, \quad i = 0,1,\ldots,n-1. \qquad (3.2.10)$$

If we set $\beta_i = 1/(n-1)$, then the activation levels a_i^h's of the Hopfield network given in Relations(3.2.10) reduces to the activation levels given in Relations(3.2.2) of network A2. It is clear that A2 network, in comparison with the Hopfield network, produces meaningful activation levels a_i's by using simpler weights. This difference along with the conclusion made in Section(3.2.3) evidently leads us to conclude that the Hopfield network is either destructive or redundant in its weights.

3.2.5 A Fault Correction Hits to the Hopfield Network

Let us find the actual fault of the Hopfield network and try to correct it. If the missing self connection of each neuron be produced with its weight as defined for the other connections, then Relation(3.2.8), for some $k = 1,2,\ldots,m$, reduces to

$$w_{ij}^{'} = x_i^k (x_j^k + \beta_i / x_j^k), \quad i = 0,1,\ldots,n-1; \quad j = 0,1,\ldots,n-1$$

Relation(3.2.9) becomes

$$snet_i^h + w_{ii}^{'} x_i^s = w_{ii}^{'} x_i^s + \sum_{\substack{j=0 \\ j\neq i}}^{n-1} w_{ij}^{'} x_j = \left(\sum_{\substack{j=0 \\ j\neq i}}^{n-1} w_{ij} x_j \right) + w_{ii}^{'} x_i^s + (n-1)x_i^s\beta_i = anet_i^{sh} + w_{ii}^{'} x_i^s + (n-1)x_i^s\beta_i$$

where $w_{ii}^{'} x_i^{'} = [x_i^s (x_i^s + \beta_i / x_i^s)]x_i^s = w_{ii}^{'} x_i^s + \beta_i x_i^s$ yields

$$snet_i^h + w_{ii}^{'} x_i^s = snet_i^h + w_{ii}^{'} x_i^s + x_i^s\beta_i = (snet_i^h + w_{ii}^{'} x_i^s) + x_i^s\beta_i = 1 + x_i^s\beta_i$$

and then Relation(3.2.10) becomes

$$a_i = snet_i^{\prime\prime} - 1 + w_{ii}x_i^{\prime} + (n-1)x_i^{\prime}\beta_i = x_i^{\prime}\beta_i + (n-1)x_i^{\prime}\beta_i = nx_i^{\prime}\beta_i = c_i x_i^{\prime} \quad (3.2.11)$$

where $c_i = n\beta_i$, $i = 0,1,....,n-1$

The activation levels given in Relation(3.2.11) is meaningful as compared to that in Relation(3.2.10). Thus it is proved that producing the missing self connections to the neurons of the hidden layer of the Hopfield network produces meaningful activation levels. To make it more meaningful let us drop the constant c_i to have

$$a_i = x_i^{\prime}, \quad i = 0,1,....,n. \quad (3.2.12)$$

But interestingly it is clear from Relation(3.2.12) that the weights and activation of the layer could not provide anything other than the sample patterns that can simply be stored in memory. This means that the Hopfield network does not qualify as a reasonable network.

3.3 Rejection of Boltzmann Machine

Since the development made in Section(3.2), is irrespective of the settlement of the Hopfield network into local minima, therefore, it is equally applicable to reject the Boltzmann Machine, as well, with the same conclusion that the Boltzmann Machine does not qualify as a reasonable network.

It may be useful to show that the weights of the trained Boltzmann Machine are the shifted versions of the weights of the Hopfield network. According to [4] the weights of Boltzmann Machine are trained by

$$\Delta W_{ij} = \in (P_{ij}^{+} - P_{ij}^{-})$$

Without going into the details of the dependency of P_{ij}^{+} and P_{ij}^{-} upon the sample patterns, we can write $\in (P_{ij}^{+} - P_{ij}^{-}) = \beta_{ij}$, and $\Delta W_{ij} = W_{ij}^{new} - W_{ij}^{old}$, so that

$$W_{ij}^{new} = W_{ij}^{old} + \beta_{ij}$$

where at time $t = 0$, the weights W_{ij}^{old} are the same as that of the Hopfield network. After successful training of the network, we will have

$$W_{ij}^{new} = W_{ij}^{old} + \sum_{t=1}^{r} \beta_{ij}^{t} = W_{ij}^{old} + c_{ij}, \text{ where } c_{ij} = \sum_{t=1}^{r} \beta_{ij}^{t}$$

This shows that the weights of the trained network are the shifted versions of the weights of the Hopfield network.

3.4 A Multiclass Linear Classifier ANN

A linear classifier ANN is developed which classifies objects of m linearly non-separable classes. The ANN is straight line activated and applies Equation(2.3.4) for its weights construction. Its design topology enable us to classify the scaled versions of known patterns as well. This ability is due to the straight line activation and dual connections of inputs with the first hidden layer. Its model is shown in Figure(3.4.1).

Topology and Activation

The number of neurons in first hidden layer are the same for each class. Practically, the number of the neurons from class to class may differ. The number of neurons for the upper and for the lower boundaries of a class may also differ.. This fact will be explained in Section(3.9). Activations and the topological details of the model are given below:

Input layer: $n+1$ neurons; 1 for bias and n for the n-dimensional pattern X; all connections are to the first hidden layer.

First hidden layer: $2mn$ total neurons; out of which mn correspond to the upper boundary and other mn correspond to the lower boundary; m is the number of classes; 2 inputs to each neuron. The weights, thresholds, and activation of the layer is the same as that of the ANN shown in Figure(2.11.2) with the only deference that here $i, j = 0,1,\ldots,n$.

Hidden layer2: $2m$ total neurons; 2 neurons for each class; n inputs to each neuron. The weights, thresholds, and activation of the layer is the same as that of the ANN shown in Figure(2.11.2) with the only deference that here $i, j = 0,1,\ldots,n$.

Output layer: m neurons; 1 neuron per class; 2 inputs to each neuron; an input is either 1 or 0; the weight of each connection of the layer is 1; threshold value of each neuron is 2. If f denote the activation function, then the activation is given as

$$f(net) = \begin{cases} 1, & \text{if } net = 2 \\ 0, & \text{otherwise} \end{cases}$$

where net denotes the net input of the neuron.

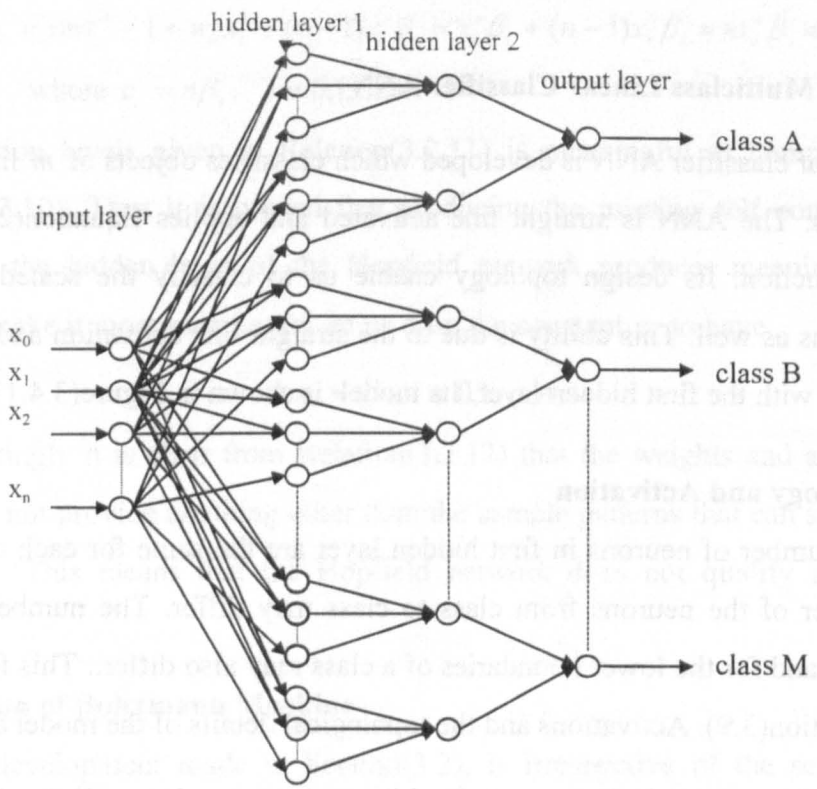


Figure 3.4.1: A Model of Multiclass Linear Classifier ANN.

3.5 A Multiclass ANN Acted upon Segment Thresholds

A classifier ANN is presented in Figure(3.5.1) which demonstrates the advantage of a straight line activation. The ANN applies the high force and the low force segment threshold decision boundaries found from the high force and the low force limiting signals $X\max$ and $X\min$. Procedure of finding segment threshold decision boundaries can be seen in Section(2.16).

First hidden layer: total $2m$ neurons; upper half of the layer corresponds to high force thresholding and the lower half corresponds to low force thresholding; Each neuron activates the straight line, for each input component, corresponding to its segment.

Weights: If y_1^k, y_2^k are the greatest most elements in kth segment of high force pattern, then weights, say w_1^k and w_2^k, are computed by using Equation(2.3.4) as

$$w_1^k = y_1^k / ((y_1^k)^2 + (y_2^k)^2); \quad w_2^k = y_2^k / ((y_1^k)^2 + (y_2^k)^2)$$

Similarly, for low force pattern, we have

$$w_1^k = z_1^k / ((z_1^k)^2 + (z_2^k)^2); \quad w_2^k = z_2^k / ((z_1^k)^2 + (z_2^k)^2)$$

Activation for *jth* actual input corresponding to high force *kth* segment:

$$net = w_1^k t_j + w_2^k x(t_j) - T; \quad f(net) = \begin{cases} 1, & \text{if } net \le 0 \\ 0, & \text{otherwise} \end{cases}$$

where t_j is such that $I_k^h : t_k \le t_j \le t_{k+1}$, I_k^h is associated with the segment.

Activation for *jth* actual input corresponding to low force *kth* segment:

$$net = w_1^k t_j + w_2^k x(t_j) - \Gamma; \quad f(net) = \begin{cases} 1, & \text{if } net \ge 0 \\ 0, & \text{otherwise} \end{cases}$$

where t_j is such that $I_k^l : t_k \le t_j \le t_{k+1}$, I_k^l is associated with the segment.

It should be noted that the lengths of the intervals I_k^h and I_k^l may by chance be the same. Activation of other layers is the same as that of given in Figure(3.4.1).

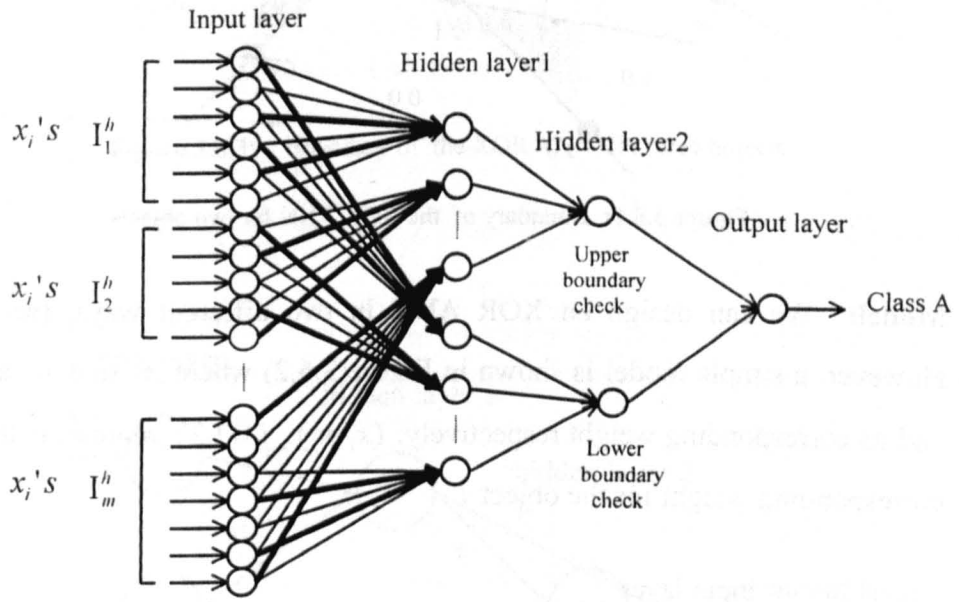


Figure 3.5.1: A Multiclass ANN acted upon thresholding boundaries.

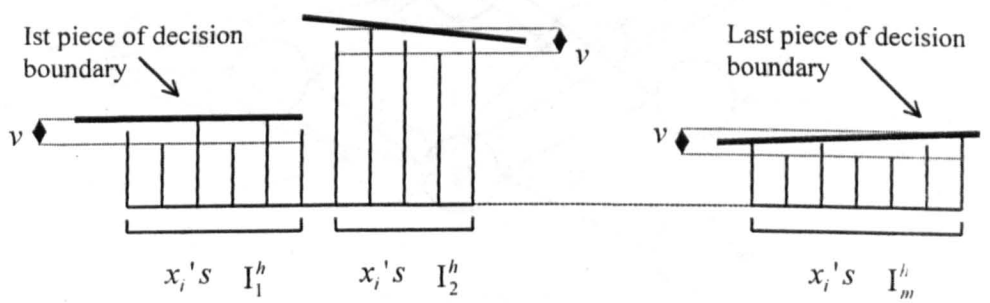


Figure 3.5.2: Partition of high force limits signal and having high force decision boundary

3.6 A New Multiclass XOR Artificial Neural Network

3.6.1 An XOR ANN for two Objects

Boundary Pattern: Consider, for example, two objects A and B such that each of the object requires only one feature (or one straight line) to be recognised and separated exclusively from each other. Consider the boundary pattern (y_0, y_1, y_2) shown in Figure(3.6.1). Note that the straight line passing through (y_0, y_1) separates the object A exclusively from object B. Similarly, The straight line passing through (y_1, y_2) separates the object B exclusively from the object A.

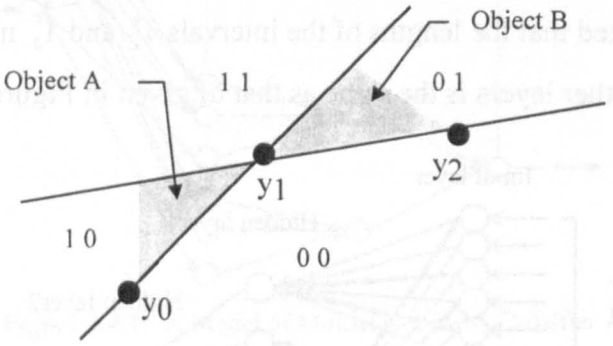


Figure 3.6.1: Boundary of the XOR ANN for two objects.

Model: We can design an XOR ANN in two different ways, (see Section(3.6.2)). However, a simple model is shown in Figure(3.6.2) where x_0 and u_0 are the bias value and its corresponding weight respectively. (x_i, u_i), $i = 1,2$; represents the feature and the corresponding weight for the object i.

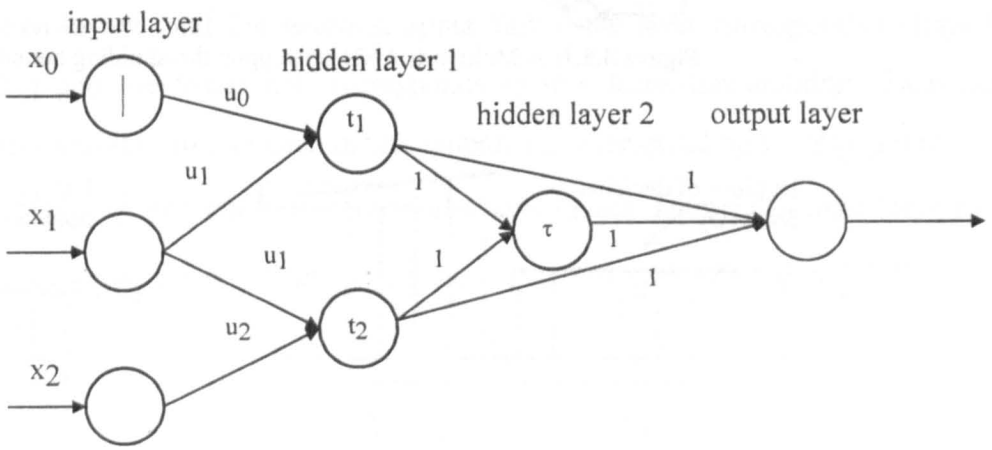


Figure 3.6.2: An XOR ANN for two objects

3.6.2 Two XOR ANNs for three Objects

Boundary Pattern: Consider three objects A, B, and C such that each object requires only one feature (or one straight line) to be recognised and separated exclusively from the others. Consider the boundary pattern (y_0, y_1, y_2, y_3) shown in Figure(3.6.3) where each object is separated exclusively from the remaining objects by a straight line.

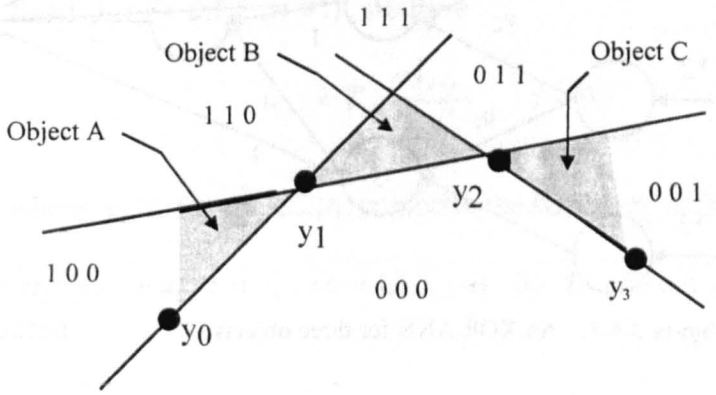


Figure 3.6.3: Boundary of the XOR ANN for three objects

Option 1:

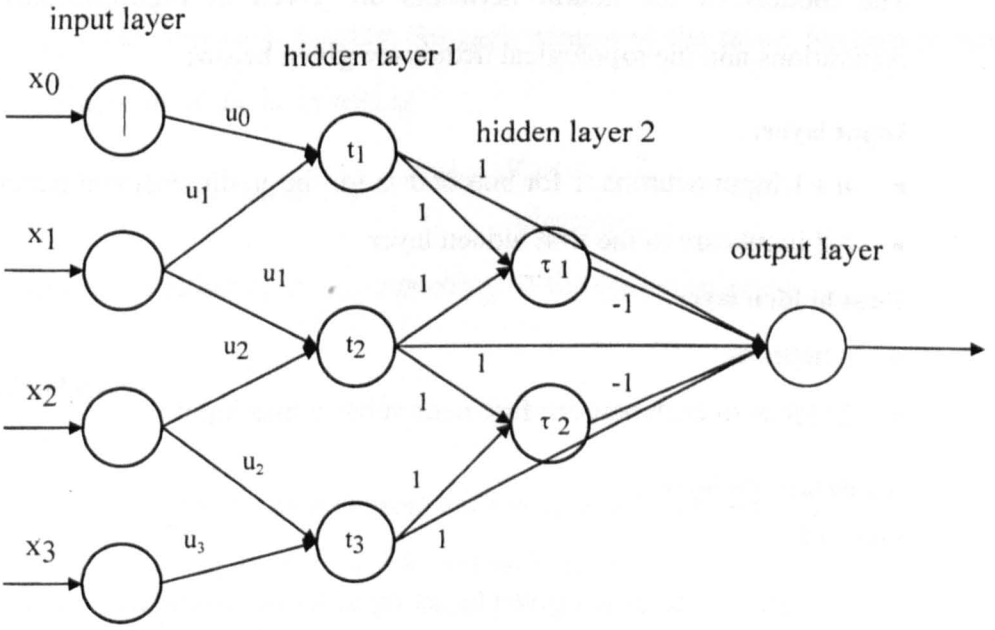


Figure 3.6.4: An XOR ANN for three objects

Option 2:

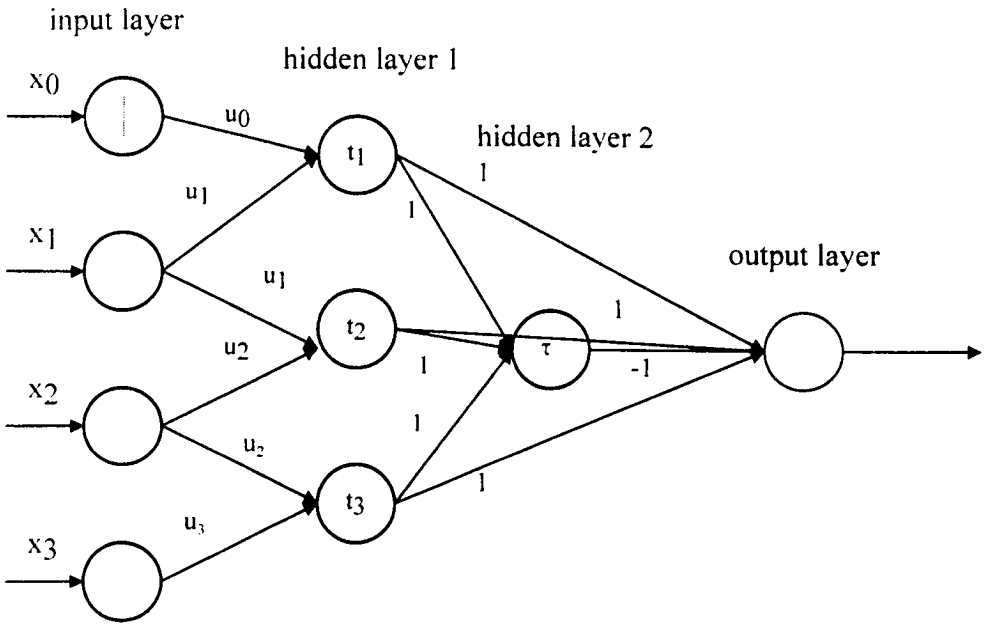


Figure 3.6.5: An XOR ANN for three objects

3.6.3 Two XOR ANNs for n Objects

Topology and activation

The models of the neural networks are given in Figures(3.6.6) and Figure(3.6.7). Activations and the topological details are given below;

Input layer:

- $n+1$ input neurons, 1 for bias and n for the n-dimensional pattern.
- All inputs are to the first hidden layer.

First hidden layer:

- n neurons.
- 2 inputs to each neuron; first neuron has a bias input.

Training of the layer:

Option 1

- If T denotes the given target input for the boundary pattern $(y_0, y_1, \ldots, y_n)$, then the weights (including the bias weight), denoted by u_i are computed as

$$u_i = Ty_i / \lambda \,,\ i = 0,1,\ldots,n\,.$$

- The threshold value t_k, for the *kth* neuron is computed as

$$t_k = u_{k-1}y_{k-1} + u_k y_k = (Ty_{k-1}^2 + Ty_k^2)/\lambda,\ \lambda = \sum_{k=1}^{n} y_k^2 .$$

Option 2

- If T_j denotes the target input (also the threshold value) for the *jth* neuron of the layer, then the weights are computed as follows:

  For every fixed j $(j = 0,1,\ldots,n-1)$, we have

$$u_{i+j} = T_{j+1}\frac{y_{i+j}}{\sum_i y_{i+j}^2},\quad i = 0,1 .$$

  where y_k's $(k = 0,1,\ldots,n)$ comprise the boundary of XOR region.

  Obviously $u_{k-1}y_{k-1} + u_k y_k = T_k$, $(k = 1,2,\ldots,n)$. So T_k can act as the threshold value for *kth* neuron.

Trained layer:

- On receiving an actual pattern $X = (x_0, x_1, \ldots, x_n)$, the layer computes net input $netl_k$, say, as $netl_k = u_{k-1}x_{k-1} + u_k x_k$, $k = 1,2,\ldots,n$.

- $h1(\alpha) = \alpha$ is the activation function for each neuron of the layer. For actual pattern X, the *kth* neuron of the layer acts as

$$h1(netl_k) = \begin{cases} |netl_k|, & \text{if } netl_k \geq T_k \\ 0, & \text{otherwise} \end{cases}$$

  where threshold is either t_k or T_k depending on the option selected.

Second hidden layer:

Option 1:

- $n-1$ neurons; n inputs to each neuron; each weight is 1; the threshold value for *kth* neuron is $netl_k$; and the activity of *kth* neuron is given by

$$net2 = \sum_{k=1}^{n-1}|netl_k|;\quad f(net2) = \begin{cases} netl_k, & \text{if } net2 > netl_k \\ 0, & \text{otherwise} \end{cases},\quad k = 1,2,\ldots,n-1$$

Option 1:

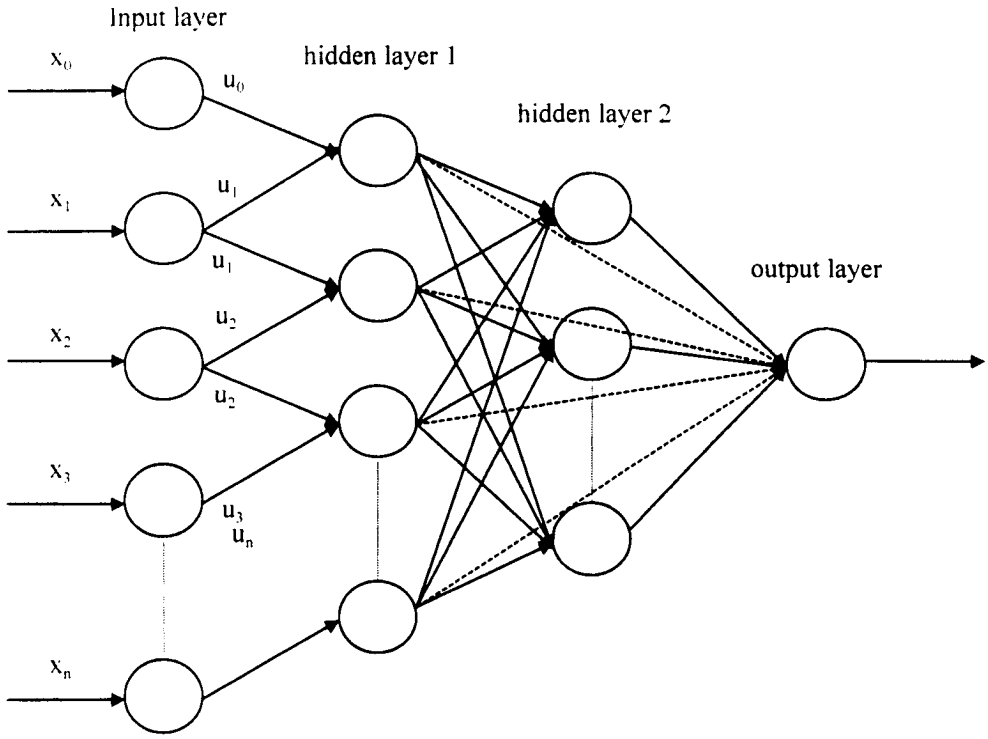


Figure 3.6.6: An XOR ANN for n objects

Option 2:

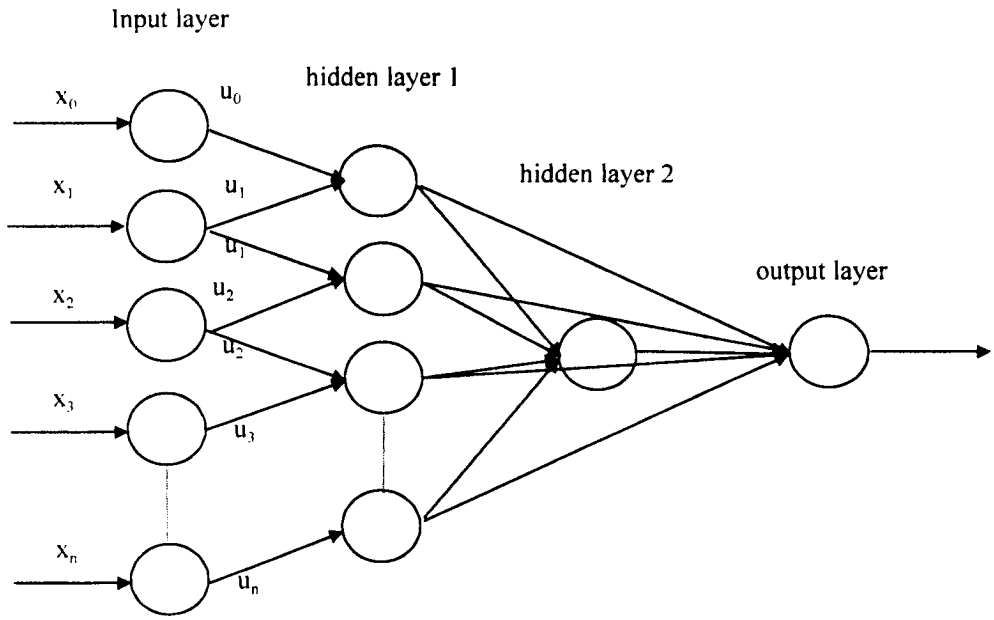


Figure 3.6.7: An XOR ANN for n objects

Option 2

- Only 1 neuron; n inputs to the neuron; each weight is 1; threshold value t, say, is computed instantly as

$$t = \min(netl_1, netl_2, ..., netl_n)$$

and finally, the activity is given by

$$net2 = \sum_{k=1}^{n-1} |netl_k| ; \quad f(net2) = \begin{cases} net2, & \text{if } net2 > t \\ 0, & \text{otherwise} \end{cases}, \quad \text{k=1,2,.....,n-1.}$$

Output layer:

- 1 output neuron; n inputs from first hidden layer with each weight equal to 1 while $n-1$ inputs from second hidden layer with each weight equal to -1.
- If out denotes the net input of the neuron then activity of the neuron is given by

$$out = \sum_{j=1}^{n} netl_j + \sum_{k=1}^{n-1} net2_k ; \quad f(out) = \begin{cases} out, & \text{if } out > 0 \\ -1, & \text{otherwise} \end{cases}$$

3.7 Training of the XOR ANN for n Objects

This method, which is being introduced here first time, extracts two elements from a training set for each class of patterns and computes their weights. The details are given below;

Step 1: Collect all the sample patterns for a class γ (γ=1,2,... n) and construct the training matrix with sample patterns as its rows.

Step 2: Find the minimum and maximum of each column and then build two patterns (say $X\min_\gamma$ and $X\max_\gamma$) from these values.

Step 3: Find two numbers ub_γ and lb_γ such that $ub_\gamma = \max(X\max_\gamma)$ and $lb_\gamma = \min(X\min_\gamma)$.

Step 4: Repeat the above three steps for all classes.

Step 5: Compute weights uw_γ and lw_γ for each class γ in the same way as were computed in Section (3.6.3).

Step 6: Arrange the elements lb_0, lb_γ's, and ub_γ's and their corresponding weights in the following form

$$lb_0,\ ub_1,\ lb_1,\ ub_2,.....,ub_{n-1},\ lb_{n-1},\ ub_n.$$
$$lw_0,\ uw_1,\ lw_1,\ uw_2,.....,uw_{n-1},\ lw_{n-1},\ uw_n.$$

Step 7: In order to minimise the complications and reducing computing time we find the intersection points (say $y_1, y_2, ..., y_n$) of each two lines described below:

(i) through lb_i and ub_{i+1}, and

(ii) through ub_j and lb_{j+1}, $j = 1,2,.....,n-1$.

Step 8: Select target inputs and threshold values and compute weights corresponding to y_i's accordingly.

3.8 A new XOR ANN as a Detector for m Objects

It is to be noted that the foregoing concept can be generalised when there are m objects such that each object requires n features or n pieces of straight lines to be recognised exclusively. Practical use of such an ANN is to detect the happening of anyone of the m events in the industrial automation, presence of anyone of the m known persons or objects, etc. Structure of this ANN can be designed by removing the input layer and the first hidden layer of one of the XOR ANNs shown in Figure(3.6.6) and Figure(3.6.7) and then combining the rest of the network with an ANN given below in (i)-(iii).

(i) the ANN shown in Figure(3.4.1); when each object is recognised with n pieces of straight lines.

(ii) the ANN shown in Figure(4.7); when each object or event is recognised with n features.

(iii) the ANN shown in Figure(5.7.1).

For example. let us combine the XOR ANN shown in Figure(3.6.6) with the ANN shown in Figure(3.4.1) to have an ANN shown in Figure(3.8.1).

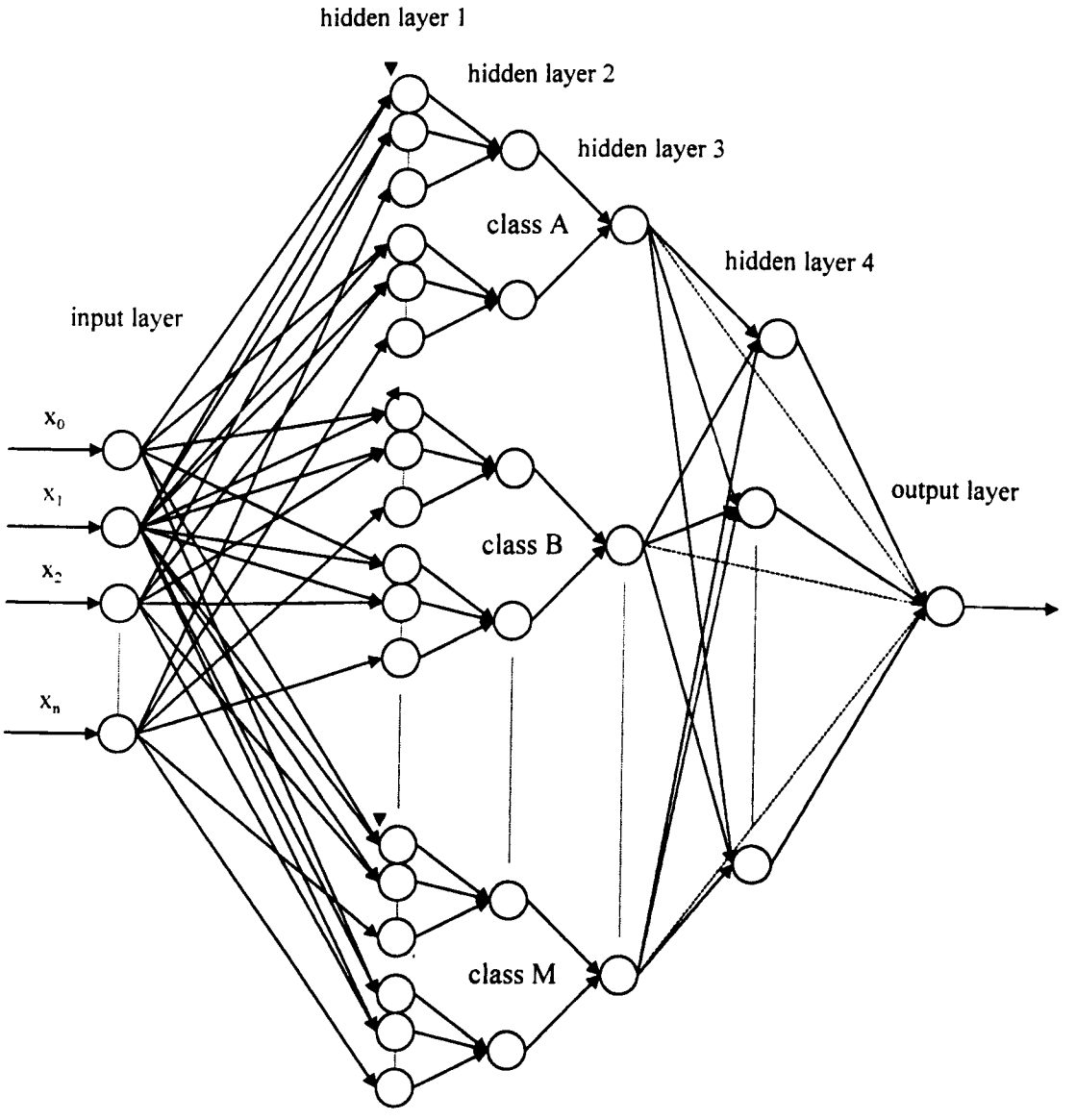


Figure 3.8.1: A model of the XOR ANN detector for m classes

3.9 A Generalised XOR ANN Detector for m Objects

There are many situations in which some of the m objects or events are recognised with different number of features. For example, if we recognise some n-dimensional signals from the number of peaks and bottoms of their cusps and troughs respectively, then obviously the number of peaks and bottoms of cusps and troughs respectively varies from signal to signal. This means that the topology of the first hidden layer will vary from object to object. Similarly, In some cases the number of high force neurons may differ from the number of low force neurons for a single object. For example, in order to recognise an object of triangular shape from its boundary we require three neurons in the first hidden layer of an ANN. Obviously there will be either one low force and two high force neurons or otherwise one high force and two low force neurons. Thus the topology of the first hidden layer is variant from object to object.

In such a situation, the number of input neurons should equals the number of high force plus the number of low force neurons of the biggest-dimensional object. Moreover, the upper most input neurons should be attached to that part of first hidden layer which corresponding to a smaller object so that the remaining input neurons be kept inactive properly.

For example, we design an XOR ANN Detector as shown in Figure(3.9.2) for three objects; a rectangular, a triangular and an hexagonal shaped objects shown in Figure(3.9.1).

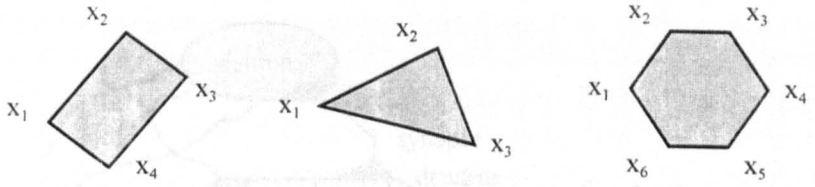


Figure 3.9.1: A rectangular, a triangular and a hexagonal object

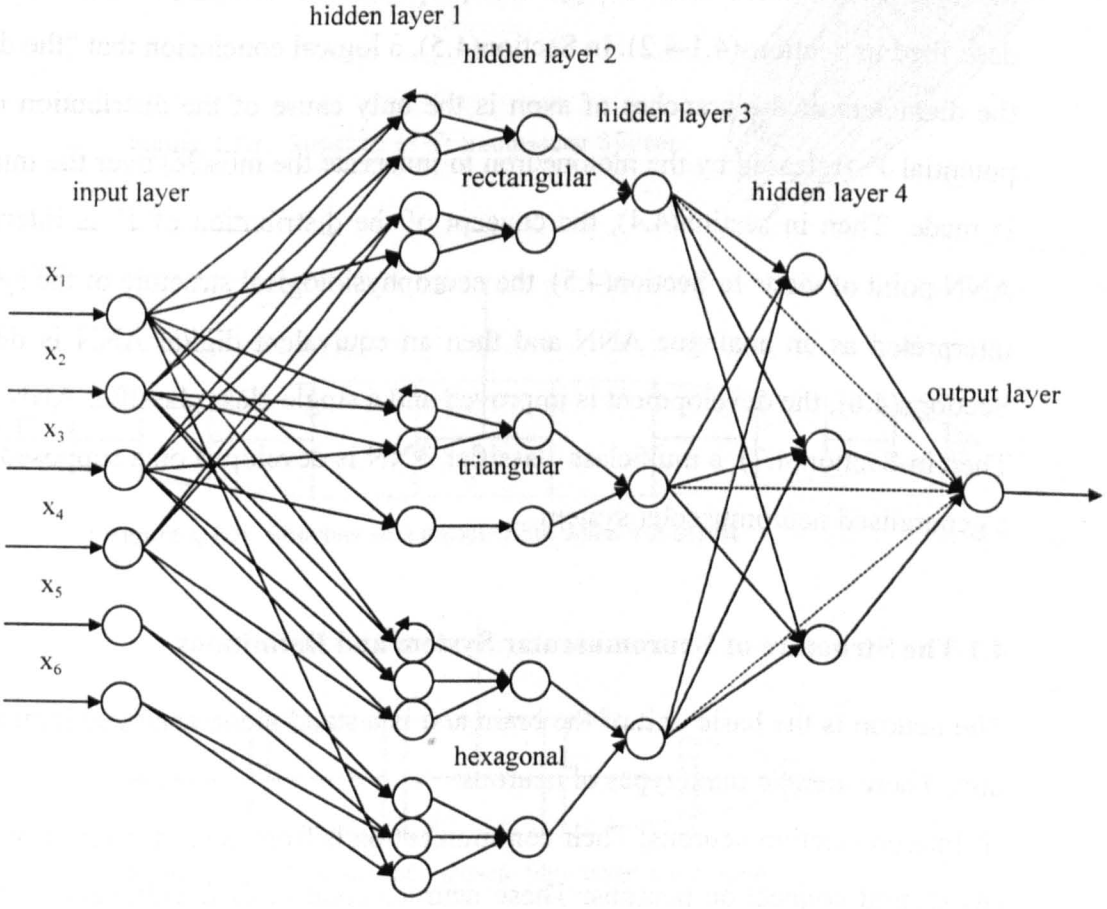


Figure 3.9.2: A model of the XOR ANN Detector for 3 object;
a rectangular, a triangular and a hexagonal shaped.

Biological Neuromuscular Systems and Artificial Neural Networks

The human brain is complicated and is poorly understood. Along with the neurophysiological research, analytical investigation is necessary, especially for the development of intelligent artificial neural networks. In this chapter the neuromuscular system is investigated. The investigation is based upon the available information about the system. Relevant terminology and properties of components of the system are described in Sections(4.1-4.2). In Section(4.5), a logical conclusion that "the difference in the diameters of the branches of axon is the only cause of the distribution of electrical potential V (released by the motoneuron to innervate the muscle) over the muscle fibres" is made. Then in section(4.4), the concept of the distribution of V is interpreted from ANN point of view. In Section(4.5), the neurophysiological structure of the system is first interpreted as an analogue ANN and then an equivalent digital ANN is developed. In Sections(4.6), the development is improved and a single class classifier ANN is achieved. Then in Section(4.7), a multiclass classifier ANN is developed on a supposed structure of a generalised neuromuscular system.

4.1 The Structure of Neuromuscular System and Definitions

The neuron is the basic unit of the brain and is a stand-alone analogue logical processing unit. There are two main types of neurons:

(i) Interconnection neurons; Their communication is from neurons to neurons.

(ii) Output connection neurons; These neurons connect different regions of the brain to each other, connect the brain to the sensory organs into brain, and connect the brain to the muscles.

A neuron accepts many inputs, which are most probably all added up in some fashion. It is believed that if enough and, most probably, relevant active inputs are received at once, then the neuron is activated and fires, otherwise remains in its inactive state[1].

We consider an output connection neuron that connects the brain to a muscle. Figure(4.1.1) shows the basic features of a neuron.

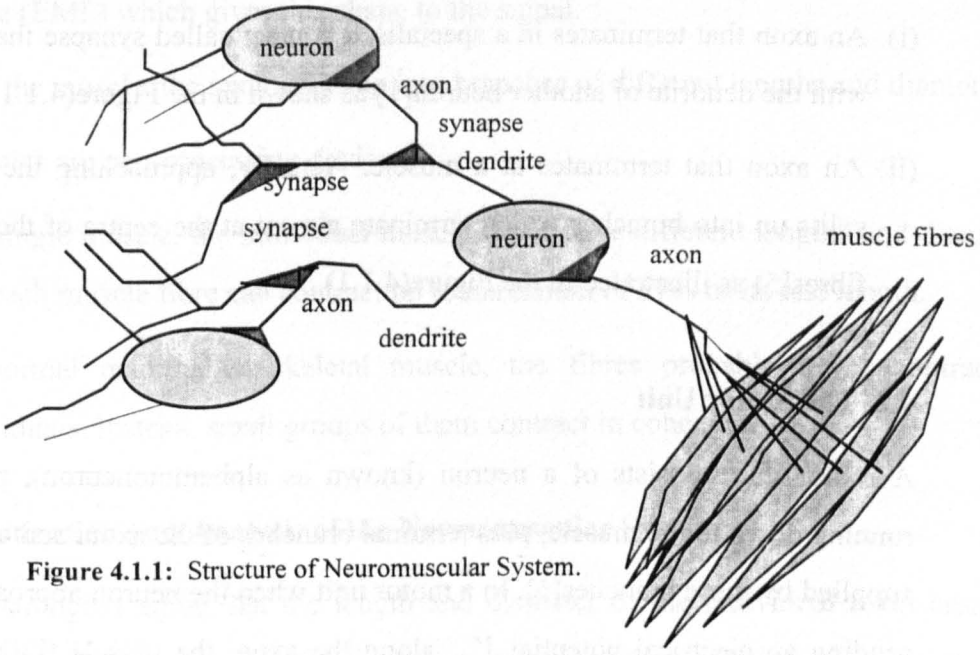


Figure 4.1.1: Structure of Neuromuscular System.

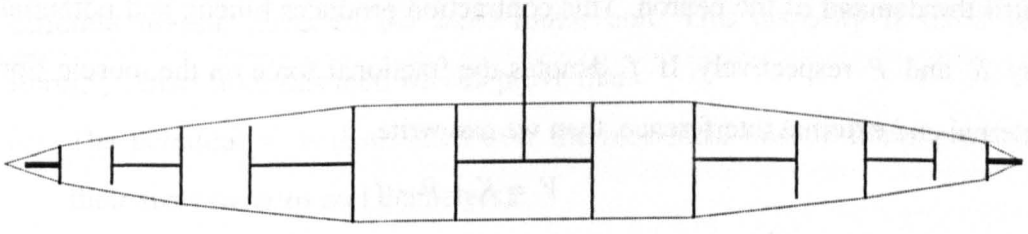


Figure 4.1.2: Structure of a muscle fibre when it is at rest.

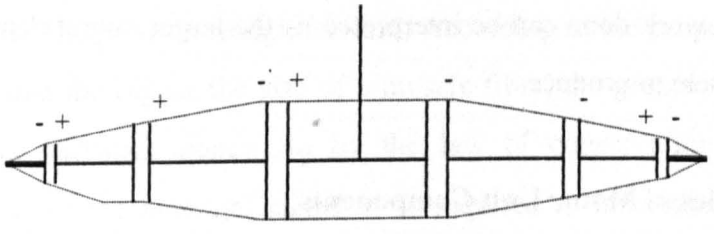


Figure 4.1.3: Structure of a muscle fibre when it is contracted.

A muscle is comprised of a number of muscle fibres of different lengths and diameters. A single muscle fibre can be interpreted as a series of capacitors as shown in Figure(4.1.2) and Figure(4.1.3). The soma is the body of the neuron. The dendrites act as the input connections through which the inputs to the neuron arrive. The axon, unlike the dendrite,

is electrically active and serves as the output channel of the neuron. There are at least two types of the axons.

(i) An axon that terminates in a specialised contact called synapse that couples the axon with the dendrite of another neuron[1] as shown in the Figure(4.1.1).

(ii) An axon that terminates at a muscle. Actually, approaching the muscle, this axon splits up into branches which terminate almost at the centre of the individual muscle fibres[5] as illustrated in the Figure(4.1.1).

4.2 The Motor Unit

A motor unit consists of a neuron (known as alphamotoneuron), plus the long axon running down to the muscle, plus terminal branches of the axon, and all the muscle fibres supplied by these branches[5]. In a motor unit when the neuron approaches the muscle by sending an electrical potential V, along the axon, the muscle fibres are contracted to fulfil the demand of the neuron. This contraction produces kinetic and potential energies say K and P respectively. If f denotes the fractional force on the muscle fibres due to internal and external interference, then we can write

$$V = K + P - f$$
$$= T \ ,$$

where T denotes the net work done by the muscle. From artificial neural network point of view, this work done can be interpreted as the target output demanded by the neuron from the muscle to produce.

4.3 Properties of Motor Unit Components

The following properties of a motor unit, which are reported in [5], are required for the analysis in Section(4.4).

(i) It can be assumed that an axon is a non-processing device. In a single motor unit there is only one axon.

(ii) An electrical signal (potential) V, having the Gaussian signal like shape, travels along the axon to the muscle. It can be assumed that it is the induced Electromotive Force (EMF) which gives this shape to the signal.

(iii) Near the muscle, the axon splits up into branches of different lengths and diameters.

(iv) Muscles are non-processing devices.

(v) In a single muscle, the individual muscle fibres have different lengths and diameters and each muscle fibre can contract up to maximum of 57% of its rest length.

(vi) In normal mammalian skeletal muscle, the fibres probably never contract as individuals. Instead, small groups of them contract in concert.

4.4 Investigation and Analysis of the Neuromuscular System

Neurophysiologists agree that the length and diameter of the individual axon branches innervating individual muscle fibres causes the disparity in the time activation of different muscle fibres of the same motor unit. This disparity is fixed for a muscle fibre[5]. Apart from this fact, we can prove that

(a) The potential V is distributed over the individual muscle fibres proportionate with their sizes (lengths and diameters).

(b) This distribution is caused by the difference in diameters of the individual branches of axons.

Proof of (a): The proof is straightforward. According to the property (v) in Section(4.3), we can say that the bigger the size of a muscle fibre, the greater it can contribute to the target output (network done). So by the law of conservation of energy, a greater component v_i of the potential signal V is required to innervate a bigger muscle fibre which in turn produces a greater contribution. On the other hand, since a muscle fibre can not contract more than 57% of its rest length, so beyond a certain limit, it would become impossible to get greater output by applying a potential component v_i, however high. After this stage, either the surplus part of the potential would be wasted or otherwise, the high voltage potential would damage the muscle fibre. This contradicts to the reality that the natural products are perfect. Thus, this fact along with the physical structure of the

motor unit and the properties of the axon lead us to conclude that the potential V is distributed over the individual muscle fibres proportionate with their sizes.

Proof of (b): Again the physical structure of the motor unit and the above mentioned facts (i)-(vi) are sufficient to conclude that the potential V released by the alphamotoneuron innervates the whole muscle and that the neuron does not release separate potentials for the individual muscle fibres. Because, if we suppose contrarily that the neuron releases different potentials for different muscle fibres, then the potentials are necessarily be sent sequentially. This requires either some sensory organs at the positions of branching of the axon near the muscle to direct the information to the concerned muscle fibre or otherwise, each muscle fibre is intelligent enough to recognise its relevant message out of many others. This contradicts to the properties (i) and (iv) listed in Section(4.3).

To see that how and by whom this distribution is done, suppose, first, that V is passed directly to each muscle fibre. Then, all the muscle fibres, being non-processing devices, will produce equal amounts of contraction say C. Obviously $T = nC$, where n is a constant to balance the mechanical resistance in the muscle fibres. This fact along with property (v) listed in Section(4.3) implies that any of larger muscle fibre in a single muscle will not contract more than 57% of the length of the smallest muscle fibre in the same muscle. This implies that the size difference of the muscle fibres is aimless. Which contradicts with the reality that the natural products are perfect. Thus our supposition is wrong.

The only chance left is that the difference in the diameters of the branches of axon causes the distribution of potential V over the muscle fibres. If this is so, then definitely the branches having larger diameters correspond to the bigger muscle fibbers (to be confirmed).

4.5 Switching from Motor Unit to Real Life Activities

The activity of neuromuscular system discussed in the previous sections leads us to model an artificial neural network on the style of neuromuscular system.

4.5.1 Analogue Neuromuscular Network

Consider a motor unit consisting of a neuron, an axon having n branches, and a muscle having n muscle fibres. The motor unit can be interpreted as an analogue neural network consisting of n input neurons, a hidden layer having n neurons, and an output layer having one neuron.

- Suppose that the muscle is rigid from one side. Then the tension on the non rigid muscle tendon produced by the muscle contraction can be interpreted as the output of the output neuron of ANN.
- The distribution of the potential V over the muscle fibres can be thought as the digitisation of V into a set $(v_1, v_2, ..., v_n)$, say, such that corresponding to a branch(here after a connection) of the axon, there is one and only one v_i that passes through this connection to the connected muscle fibre.
- By ignoring the fractional force f, the potential V can be replaced by T. Where as v_i's replaced by T_i's are the target work done of the individual muscle fibres. The collection of all the muscle fibres can be thought of a hidden layer with target output of ith neuron equals T_i.

Weights and threshold values:

In a motor unit, as the target energy is distributed to the individual muscle fibres by the branches of the axon. The distribution of the target output can be defined either from the difference of diameters of the axon branches, or equivalently, from the differences in the rest lengths of the individual muscle fibres. This idea of defining weights has been presented in Section(2.3). For straight line activation and threshold boundaries, see Section(2.4) through Section(2.11), Section(2.14, and Section(2.16).

When a maximum possible potential V (that can be put across the axon) is applied, the maximum contraction pattern $(y_1, y_2, ..., y_n)$ is achieved. If d_i denotes the diameter of the ith branch of the axon, then the maximum potential T_i, that can be put across the ith branch of the axon can be computed as

$T_i = Vd_i^2 / \sum d_i^2$. or equivalently $T_i = Vy_i^2 / \sum y_i^2$. or equivalently $T_i = Vs_i^2 / \sum s_i^2$. where s_i denotes the rest size of the *ith* muscle fibre.

Obviously $d_i^2 / \sum d_i^2 = y_i^2 / \sum y_i^2 = s_i^2 / \sum s_i^2 = p_i$, where p_i is the proportionate value by which the *ith* muscle fibre contributes to the target net work done. Hence $T_i = Vp_i$. The model of the ANN is shown in Figure(4.5.1).

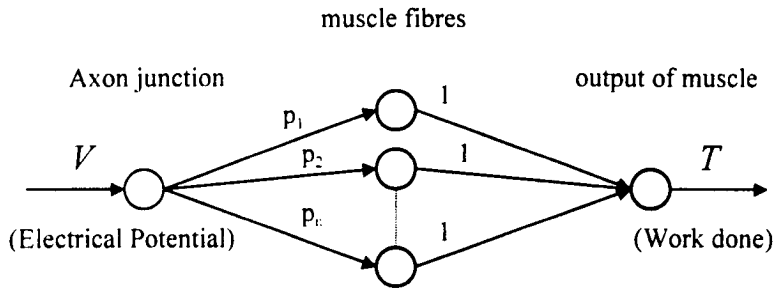


Figure 4.5.1: Model of the Analogue Neuromuscular Network

4.5.2 Digital Artificial Neuromuscular Network

If instead of the input signal V, we present a pattern $(x_1,x_2,.....,x_n)$ to the network, then the thresholds T_i and weights w_i are given by

$$T_i = (Ty_i / \sum y_i^2)y_i = w_i y_i, \quad \text{where } w_i = Ty_i / \sum y_i^2 .$$

Where y_i's comprise the upper boundary of the class of the patterns $(x_1,x_2,.....,x_n)$. The model of network is shown in Figure(4.5.2).

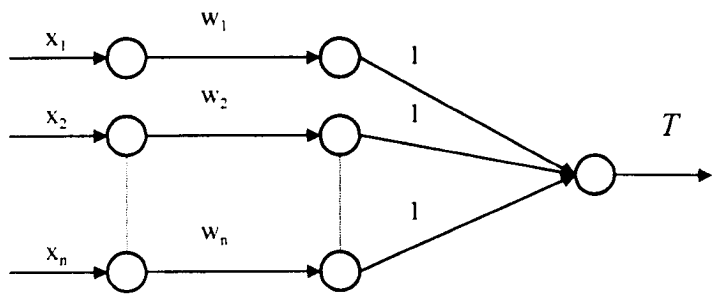


Figure 4.5.2: Model of the digital neuromuscular system

- The identity function $f(\alpha) = \alpha$ is the activation function for each neuron of both the hidden and the output layers.

Hidden layer: $f(w_i x_i) = w_i x_i$ when $w_i x_i \leq T_i$ otherwise $f(w_i x_i) = 0$

Output layer: For $T = \sum T_i = \sum w_i x_i$,

$$f(T) = T \quad \text{when } T > 0 \quad \text{otherwise } f(T) = 0$$

Note that only high force thresholds (upper bounds of the inputs) are used in the hidden layer. While the low force thresholds (lower bounds of the inputs) are implemented automatically through the activation of the output neuron. This reduces the hidden layer to half size. But for real problems, we need to check for the lower boundaries as well. That is, we need low force thresholds as well. This will be explained in the next section.

4.6 Generalisation of the Concept

In order to meet the objectives of Artificial Neural Networks, it is necessary to improve and generalise the concept of the activation of the digital neuromuscular system discussed in the previous section and then apply them to real problem.

The contraction value x_i, of the *ith* muscle fibre of a muscle can be thought of the *ith* element of a pattern. So a pattern $X = (x_1, x_2, ..., x_n)$ represents one particular contraction of the muscle of a particular pattern. A class Φ of all possible patterns then covers the whole range of the contraction of the muscle. The class being the representative for muscle contraction is bounded. That is, $\forall\ X \in \Phi$ we can find signals $X\min$ and $X\max$ such that for every $x_i \in X$, there exist $\alpha_i \in X\min$ and $\beta_i \in X\max$ such that

$$0 < \alpha_i \leq x_i \leq \beta_i\ ,\quad \text{i=1,2,.....,n} \qquad (4.6.1)$$

Moreover, in order to apply the network in practical problems, we need to shift the centroid of the muscle signal from first quadrant to elsewhere in the plane. This is equivalent to replacing [0, 57%l$_i$] (see (v) in Section 4.3) by an arbitrary interval $[\alpha_i, \beta_i]$. Thus the *ith* element of a pattern of the class can have any value from this interval. This is illustrated in Figure(4.6.1).

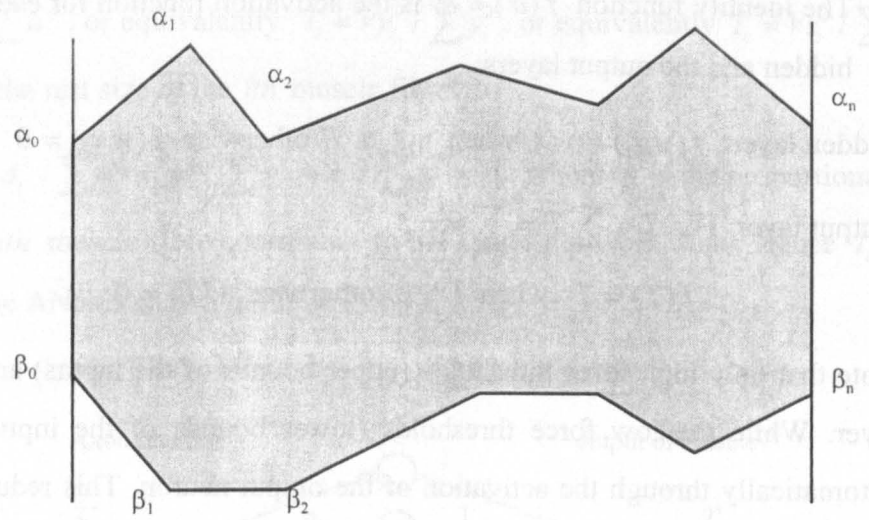


Figure 4.6.1: A class of patterns.

Thus, the modified version of relation (4.6.1) can be written as

$$\alpha_i \leq x_i \leq \beta_i, \quad \text{i=1,2,.....,n} \tag{4.6.2}$$

where α_i and β_i are the least upper bound and the greatest lower bound respectively of x_i.

We compute Γ_i and t_i, the high force threshold value and the low force threshold value, respectively, of *ith* neuron of the hidden layer as

$$\Gamma_i = T\alpha_i^2 / \sum \alpha_i^2 = u_i \alpha_i, \text{ where } u_i = T\alpha_i / \sum \alpha_i^2$$

$$\tau_i = T\beta_i^2 / \sum \beta_i^2 = v_i \beta_i, \text{ where } v_i = T\beta_i / \sum \beta\alpha_i^2$$

The model of the network is shown in Figure(4.6.2).

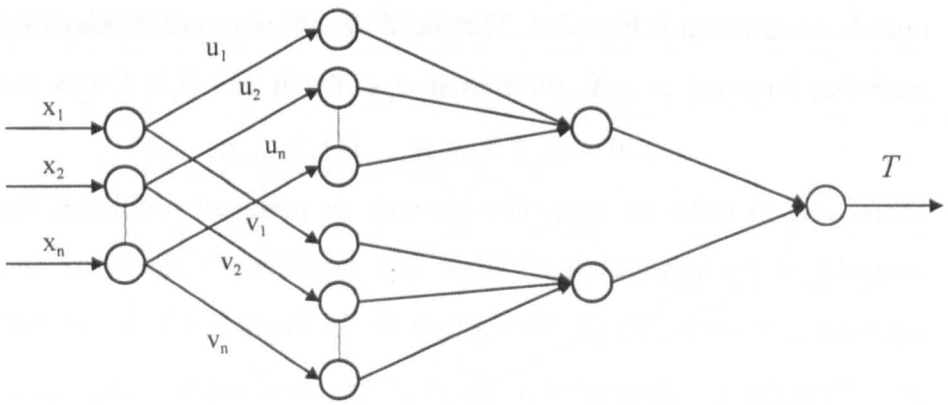


Figure 4.6.2: Model of the Neuromuscular ANN

4.7 Multimuscle Systems and an ANN

To generalise the idea, consider a motor unit consisting of one innervating neuron, one axon having mn branches, and m muscles each having n muscle fibres. The motor can be interpreted as a three layer ANN consisting of n input neurons, first hidden layer having mn neurons, second hidden layer having $2n$ neurons and an output layer having m neurons. The model of the network is given in the Figure(4.7).

This ANN is a natural example to act upon the high force threshold limits and the low force threshold limits of patterns. However, if we replace its design topology with the topology of the ANN shown in Figure(3.4.1) and act upon the high force and the low force threshold boundaries, as is the case of the ANN given in Figure(3.5.1), then the resulting ANN will be able to classify the scaled versions of samples as a known patterns.

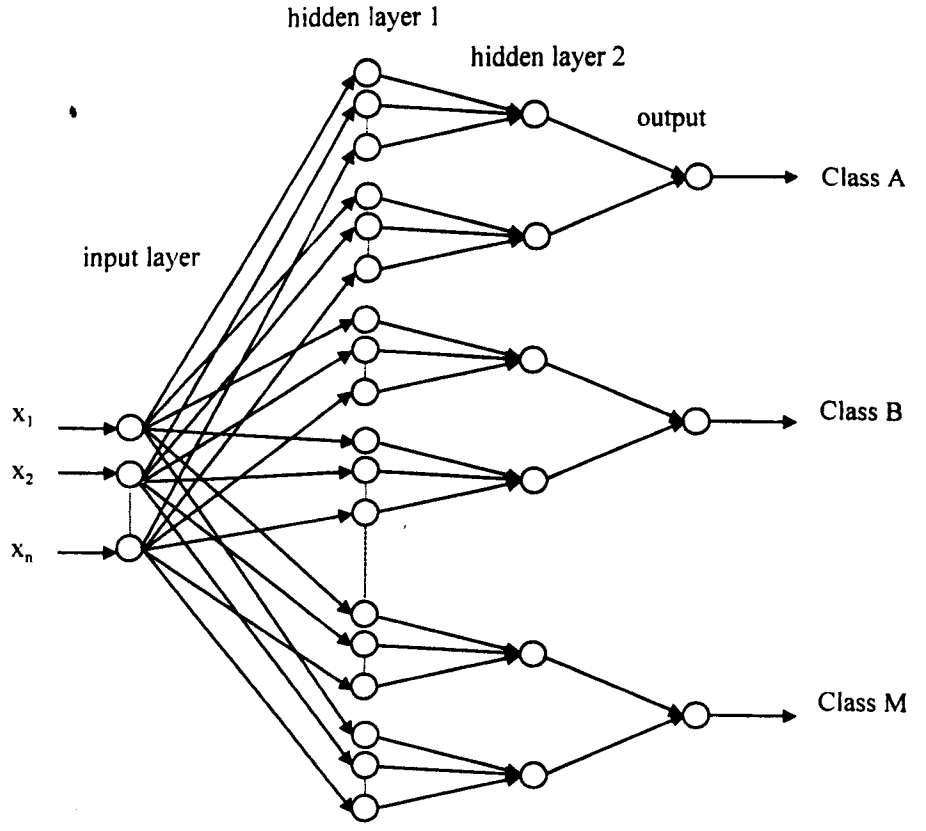


Figure 4.7: Multimuscle Neuromuscular Artificial Neural Network

Artificial Neural Networks for Image Processing

A digital image can be considered as $n \times n$ matrix of grey levels. There may be some other ways for pattern recognition but analysing its individual rows and columns seems a better one because the rows and columns when considered as signals have some wave form. So, we can extract sufficient number of features from an image data to recognise it. Some of the features are peaks of cusps and bottoms of troughs, indices of the peaks and bottoms, amplitude differences of adjacent peaks and bottoms, relative distances between adjacent peaks and bottoms, total number of cusps and troughs, ordering structure of cusps and troughs, etc. Some techniques for features detection and some ANNs showing their advantages will be presented in this chapter. Particularly, in Section(5.8), a remarkable ANN classifier is presented which is able to detect and reconstruct an input pattern as a known or an unknown one. The ANN then classifies an unknown pattern into three categories; a scaled version, or a mirror image, or a noisy version of the sample pattern. Moreover, the ANN is capable to tell us whether the noise is remove able or not. This ANN is presented as an example for pattern recognition and for usefulness of the amplitude differences of adjacent peaks and bottoms. Advances of this chapter are presented in Chapter10.

5.1 Locating Peaks of Cusps and Bottoms of Troughs

1. There is some association between cusps of rows and cusps of columns of an image. The same is true for troughs. If we correlate cusps of rows with the cusps of columns of the same image, we can find a correspondence of certain shape. But since the columns of an image are nothing more than the same data of rows seen from another angle (i.e. a right angle), so there is no need to analyse columns after having analysed rows in detail and vice versa.

2. For implementation purposes, we can draw a straight line parallel to the time axes and passing through the grey level (amplitude) 127. This line can be thought as a base line of the graph of a row. Then, relative to this line, we can calculate heights and depths

of cusps and troughs respectively. Whereas labelling cusps and troughs can be achieved by noting changes of sign of the first order derivative of the signal. That is, if $f(x)$ denotes a signal then a grey level x_0 represents the top of a cusp if $f'(x)$ changes its sign from positive to negative going from lower to higher values of x in the neighbourhood of x_0 and it represents the bottom of a trough if $f'(x)$ changes its sign from negative to positive.

5.2 Variation of Features in Different Situations.

The indices and amplitudes of peaks and bottoms, and hence their relative distances, are non-consistent with respect to dimensional variation and brightness of images. Therefore, the use of these features alone can not be recommended to train neural networks. Otherwise, the neural networks might not work effectively. Some of the possible situations will be discussed in detail.

5.2.1 A Still Image and its Features

The easiest case of image recognition is one in which the brightness and the dimensions of the image remains the same during both the training phase and the working phase of the ANN. In this case , the amplitudes and the indices of peaks and bottoms remains the same enabling us developing and implementing software/hardware systems easily.

5.2.2 Luminance and Variation of Features

The variation of luminance causes non-consistency in the amplitude spectrum of the image. The brighter the image the bigger is the amplitude. Since it is almost impossible to control the brightness of the images at the receiving end. Therefore, this fact forces us to not to consider the amplitudes alone. Otherwise an ANN trained for a brighter image may not be able to recognise a darker version of the same image.

5.2.3 Telescopic Image and Variation of Features

The absolute and the relative distances of cusp and troughs, being explicit functions of the dimension of image, are variant with respect to image expansion. This fact is important to avoid because, we can not guarantee to receive the same size of the image

for which we have trained our ANN. Moreover, we like to recognise and observe telescopic versions of images. So, the ordering structure of cusps and troughs is essential.

5.3 Data Reduction

Actual issue of this section is data reduction. The topic of noise reduction and data compression will be considered in detail in Chapters(7-9), however we need noise free data, at least, for the following purposes;

(i) Data Reduction,

(ii) Data Transmission,

(iii) Pattern Recognition,

(iv) Data Reconstruction.

We can minimise processing time, manufacturing cost, and difficulties of implementation by compressing and reducing image data. The data can be reduced with the guarantee of reconstruction.

Procedure:

In the graph of a row of an image, there are peaks of different sizes. Some of the peaks of small sizes can be ignored from the rows by declaring them noise. The rest of the peaks are of particular interest to extract some prominent features associated with them. Moreover, the possible fluctuation of data in the neighbourhood of peaks of cusps and bottoms of troughs can be discarded. We locate and record the amplitudes and the corresponding indices of the prominent peaks and bottoms. We can save these values in a three dimensional **data file of prominent features;** D1-RowNum, D2-Amplitude of peak(bottom), D3-Index of peak(bottom)

Reconstruction:

An image can be reconstructed from the data file of prominent features in two ways;

(i) by using the Wavelet Transform developed in Section(8.6),

(ii) by inserting geometric means between each pair of consecutive prominent peak and bottom of each row.

The difference between the indices of the two consecutive peaks gives width of the wavelet or/and number of geometric means to be inserted.

A piece of software code can be developed to perform this task. The flow chart of the suggested program is shown in Figure(5.3.1).

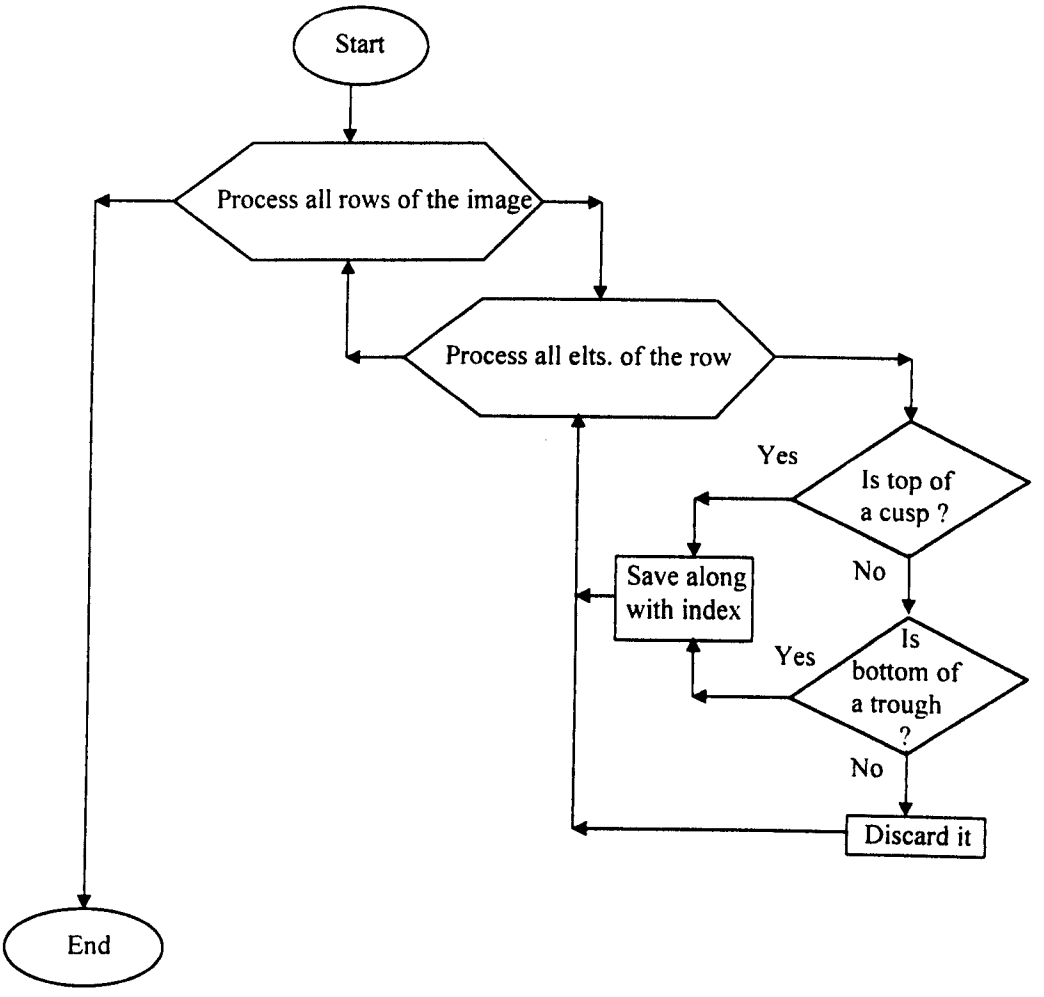


Figure 5.3.1: Flowchart to locate peaks of cusps and bottoms of troughs.

5.4 Designing ANN as an Image Classifier

An Artificial Neural Network can be designed for pattern recognition of images. A properly designed ANN can be trained for the matrices of the prominent features of many images. Moreover, the procedure of data compression discussed in the above section can be included as a pre-processor in the suggested ANN.

We can design an artificial neural network, as given in Figure(5.4.1), to classify images by their features. Note that in the figure, the box shaped unit enclosing "Image" represents the contents of the image as an input of the network. The circle shaped unit enclosing "Extract features" represents a software/hardware processing unit that extracts some required features of the image as explained in Section(5.3). These features are then passed to the diamond shaped units called comparaters. Each comparater is trained and corresponds to a class of images. After receiving features, it compares them with its inherited corresponding values and outputs the result accordingly.

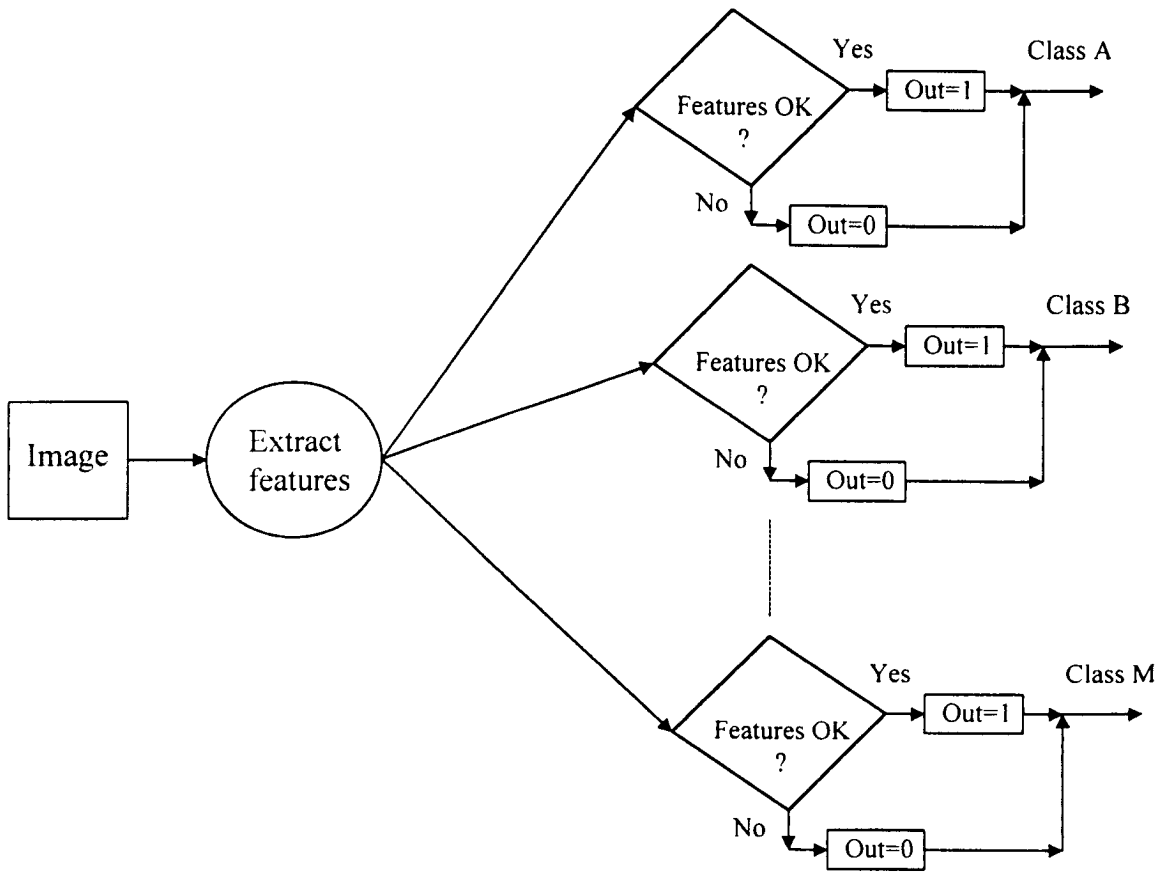


Figure 5.4.1: Basic Structure of a Classifier for Images.

It is to be noted that by following the reconstruction procedure explained in Section(5.3), a reconstruction unit can be developed and included in this artificial neural network as an option.

5.5 Training Procedure

For training, we proceed as follows:

(i) Collect sample set of images.

(ii) Determine number of peaks and bottoms in each row of each image separately.

(iii) To each neuron of hidden layer3, assign corresponding number of peaks and bottoms.

5.6 Advanced Model and its Training Procedure

Suppose that the neural network given in Figure(5.4.1) has been trained to recognise some images. Then in future, if we pass to the network a

1. **digitised version of a telescopic vision, however large or small, of one of the images:** the neural network will recognise it as the same image for which it was trained. And it will not be able to tell the size of the image. This fact can be achieved by including indices of the peaks and bottoms as the additional features.
2. **a brighter or a darker version of one of the images:** the neural network will recognise it as the same version of the image for which it was trained. Also it will not be able to tell any thing about the degree of brightness. However, This fact can be achieved by including amplitudes of the peaks as the additional features. The advanced model of ANN is given in Figure(5.7.1).

Advanced training steps are:

(i) Follow the steps (i)-(iii) of the training procedure given in Section(5.5);

(ii) Detect prominent peaks and bottoms and pick their indices.

(iii) Pick amplitudes of the prominent peaks and bottoms.

(iv) To each neuron of hidden layer2, assign corresponding number of cusps, number of troughs, peaks, indices and amplitudes of the peaks and bottoms.

5.7 Topology and Activation

The model of the neural network is shown in Figure(5.7.1). Activation and topological details are given below:

Input layer

- n neurons.
- one row is passed at one input connection.
- No activation. That is, all the input rows are directly passed to the hidden layer 1.
- All connections are to the hidden layer 1.

First hidden layer

- n neurons;
- each neuron computes number of cusps and the number of troughs of the corresponding input row data.
- For advanced model, along with the number of cusps and troughs, the neurons also detect their peaks and bottoms, picks indices and amplitudes of the peaks and bottoms.
- All connections to the hidden layer3.

Second hidden layer

- $2mn$ neurons arranged in pairs; where m stands for the number of classes of images.
- First neuron of each pair receives upper features; number of cusps, their peaks, indices and amplitudes of the peaks.
- While the second neuron of each pair receives lower features; number of troughs, their bottoms, indices and amplitudes of the bottoms.
- After receiving features, each neuron compares them with the corresponding features inherited by them during training and computes results according to situation.

Third hidden layer

- mn neurons; n neurons per class.
- two inputs to each neuron.
- Activation of each neuron is to perform logical AND.

Output layer

- m neurons; 1 neuron per class.
- n inputs to each neuron;
- An input is either 1 or 0;
- Activation of each neuron is to perform logical AND.

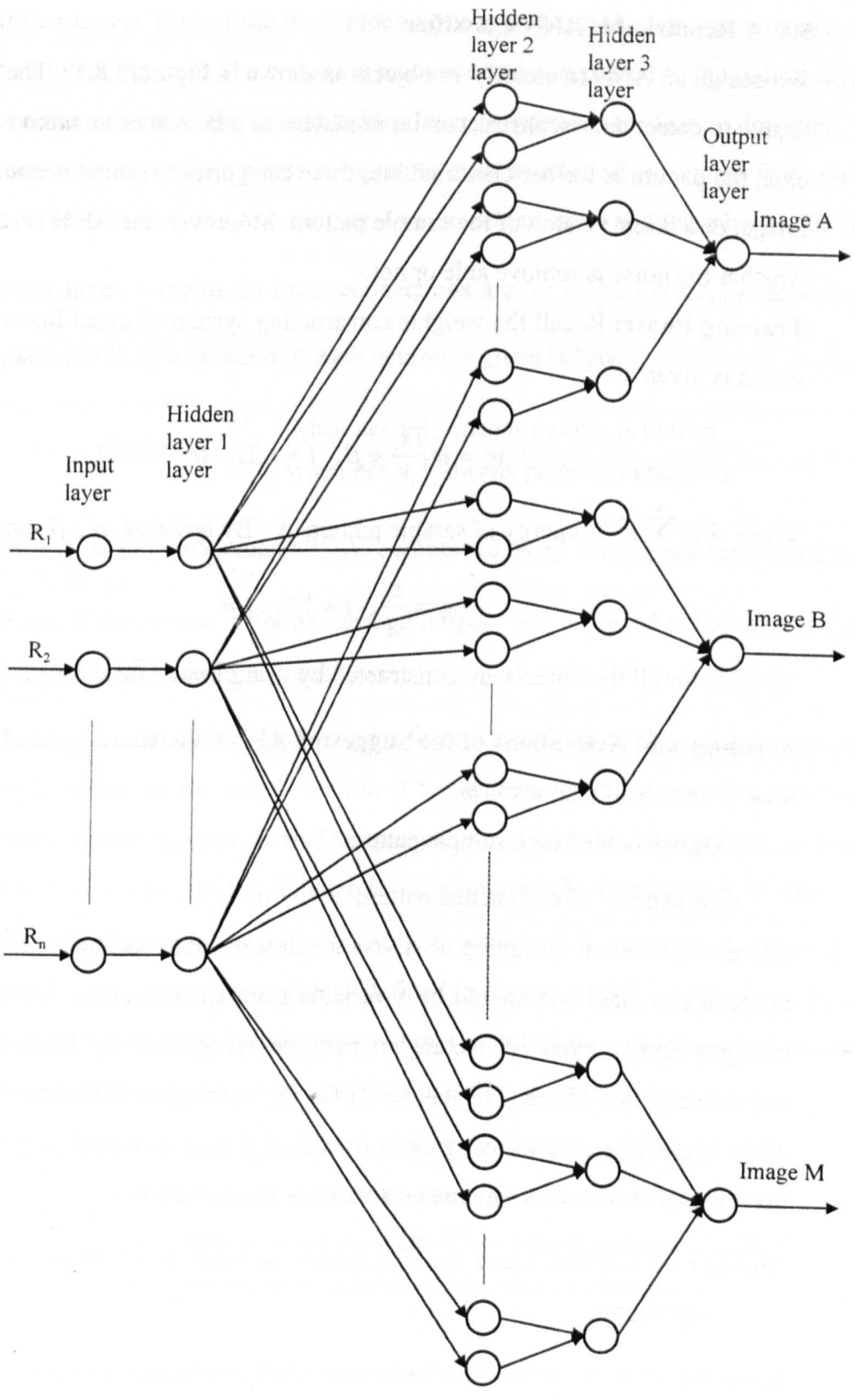


Figure 5.7.1: Image Recognition Artificial Neural Network.

5.8 A Remarkable ANN Classifier

We design an ANN to classify m objects as shown in Figure(5.8.1). The trained classifier is able to detect and reconstruct an input pattern as a known or an unknown one. In later case. the pattern is further classified into three categories; a scaled version, or a mirror image, or a noisy version of the sample pattern. Moreover, the ANN is capable to tell us whether the noise is remove able or not.

Training Phase: Recall the weights constructing system of exact linear Equations(2.3.4) which is given as:

$$w_i = \alpha \frac{Tx_i}{\lambda} + \beta, \quad i = 1,2,....,n$$

where $\lambda = \sum_{i=1}^{n} x_i^2$ is energy of sample pattern X. By ignoring α, β, and T, we have

$$w_i = \frac{x_i}{\lambda}, \quad i = 1,2,....,n \tag{5.8.1}$$

Weights for all the objects are constructed by using Equations(5.8.1).

Topology and Activations of the Suggested ANN Classifier:

Size: Number of Classes = m

Dimension of each sample pattern X = n

Dimension of each actual pattern X^a = n

However, in circuit designing of ANNs the dimension of sample patterns should be fixed for each one class and should be a variable from class to class. For example, a class of triangles and a class of rectangles may be recognised by three and four features, respectively. See Figures(3.9.1)-(3.9.2) for the topological difference in such a case. An ANN should be able to recognise a fractional part of an actual pattern. This means that the dimension of actual patterns be a variable for each class.

Number of Layers: There are five layers; an input layer; three hidden layers; and an output layer.

Input layer: Total n neurons; each neuron has one output; all connections are to the first and second hidden layers; no activation, that is the inputs are simply passed to output connections.

First hidden layer: Task of the first hidden layer is to detect the input pattern as a known or an unknown one; The pattern is reconstructed and passed to the output layer if it is known otherwise the second hidden layer is activated for feature detection. There is one neuron for each class; n inputs to each neuron; all inputs are from the input layer; each neuron has two outputs - one to output layer and the other to the second hidden layer; weights on input connections are $w_i = \frac{x_i}{\lambda}$, $i = 1,2,...n$ as defined in Equation(5.8.1); activation of each neuron is given below:

$$f(net) = \begin{cases} X^a, & \text{when } net = 1; \text{ means pattern is known} \\ -1, & \text{when } net \neq 1; \text{ means pattern is unknown} \end{cases}$$

where $net = \sum_{i=1}^{n} w_i x_i^a$; the elements of X^a, computed as $x_k^a = \lambda w_k$, and are passed to the output neuron if the pattern is known; second output '-1' is passed to the second hidden layer to activate it if the pattern is unknown;.

Second hidden layer: This layer performs feature detection of the unknown pattern. A binary bit is passed to the output neuron if the pattern be found a scaled version or a mirror image (either scaled) of the sample pattern; otherwise, the pattern is noisy or purely new. In that case, the third hidden layer is activated by passing it δ_k's. Here is one neuron for each class and should be consider a packet of n neurons; n inputs to each neuron from input layer with weights as defined in Equations(5.8.1); one input from first hidden layer with weight 1; There are two output connections - one to output neuron and other to third hidden layer. Activation of *kth* neuron is given below:

$$f(r_k, \delta_k) = \begin{cases} 1, & \text{when } r_k > 1 \ \forall\ k; \text{means } X^a \text{ is scaled up vesion of } X; \\ 0, & \text{when } 0 < r_k < 1 \ \forall\ k; \text{means } X^a \text{ is scaled down vesion of } X; \\ -1, & \text{when } r_k < 0 \ \forall\ k; \text{means } X^a \text{ is a miror image (scaled) of } X; \\ \delta_k, & \text{otherwise } \forall\ k; \text{means } X^a \text{ is noisy or purely new;} \end{cases}$$

where

$$r_k = \frac{x_k^a}{x_k} = \frac{x_k^a}{\lambda w_k} \qquad , k = 1,2,...n$$

$$\delta_k = abs(x_k^a - x_k) = abs(x_k^a - \lambda w_k)$$

Third hidden layer: The task of this layer is to decide whether the unknown pattern X^a is a noisy version of the sample pattern X or it is purely new. There is one neuron per class; one input line that comes from the second hidden layer and one output line that goes to the output neuron; weight on each input connection of the layer is 1; activation of the layer is given below:

$$f(\delta_k) = \begin{cases} 1, & \text{when } \delta_k \le \varepsilon \; \forall \; k; \text{means } X^a \text{ is noisy vesion of X and noise is removabe;} \\ 0, & \text{otherwise; means } X^a \text{ is purely new or corrupted version of X;} \end{cases}$$

where $\varepsilon = \lambda|p-b|$, p and b are the amplitudes of the peak and the bottom in between of which x_k^a lies and λ is a select able constant.

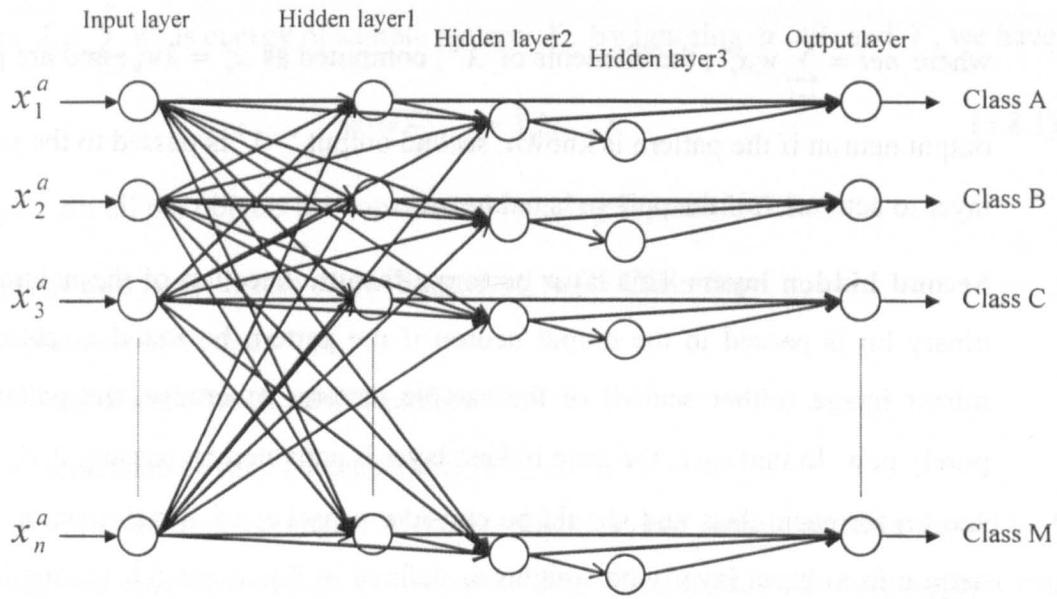


Figure 5.8.1: Basic Structure of an ANN Classifier Comparable with the Hopfield ANN.

Software: A software program **tremann.cpp** (Testing the logic of REMark able ANN) is developed and listed in Appendix(3.5). For this program, a subroutine is developed which finds the peaks of cusps and bottoms of troughs and then finds the difference of amplitudes between each couple of adjacent peak/bottom or bottom/peak. The purpose is to decide whether a noisy unknown pattern qualifies as a noisy version of the sample pattern or not.

Note: A number of ANNs and their software checks based upon the theory of this chapter will be presented in Chapter10.

From Fourier Analysis to Wavelet Analysis

This large chapter can be divided into five groups. **Group-1:** In the first two sections, the Fourier transform and the Short Time Fourier Transform (STFT) is elaborated. Their important feature are checked by developing the software with graphical outputs. In Section(6.3), a computing redundancy in STFT is expressed and a procedure to remove it is presented. The development is verified and demonstrated with support of graphical outputs of the software developed. **Group-2:** After listing some drawbacks of STFT, in Section(6.5) the relative frequency analysis is elaborated. The analysis is demonstrated with graphical support of the software programs developed. In the end of this group, the switching from STFT to Wavelet Transform is elaborated. Three software programs has been developed to demonstrate the functioning of the Wavelet Transform. **Group-3:** After expressing some basics of Wavelet Decomposition, an important development is made in Section(6.11). It is shown that a real time signal can be decomposed in terms of particular six Gaussian Wavelets. The procedure of development and reconstruction is checked against the software programs. There are six programs developed and their graphical outputs are included in the text. **Group-4:** This group consists of three sections. The Section(6.12) has the honour to associate two wavelet transforms, the DWT and the CWT, with the wavelet decomposition introduced in Section(6.11). This development can be said the height of the chapter. The presentation is analytical and is checked against the software. In Section(6.13) some larger wavelet packets are developed by utilising the Gaussian wavelet packet of six windows. In Section(6.14), the Gaussian wavelet packet is transformed to a binary wavelet packet which provides us orthonormal wavelet basis while preserving its application for digitised CWT for noise reduction and data compression. Its performance is checked. **Group-5:** The purpose is to make the things easier for the subsequent chapters, particularly, for Chapters(7-9). Some useful refinement equations has been interpreted and elaborated mathematically. The developments are presented with graphical support.

6.1 Fourier Transform and its Drawbacks

Let $L^2(0,2\pi)$ denotes the space of 2π-periodic square-integrable functions. The Fourier transform (FT) of a function $f(t) \in L^2(0,2\pi)$, denoted by $\hat{f}(\omega)$, is defined as

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \tag{6.1.1}$$

whereas the inverse Fourier transform (IFT)is defined as

$$f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega t} d\omega, \tag{6.1.2}$$

Generally, a function is square-integrable if the integral of its square exists. If a square-integrable function is defined on the whole real-axis then it is said to belong to $L^2(\Re)$.

Software 6.1: The Fourier Transforms defined in Equations (6.1.1) and (6.1.2) are tested by writing the program named **tfft1d.cpp** listed in Appendix(1.1). The performance of the program is shown in Figure (6.1).

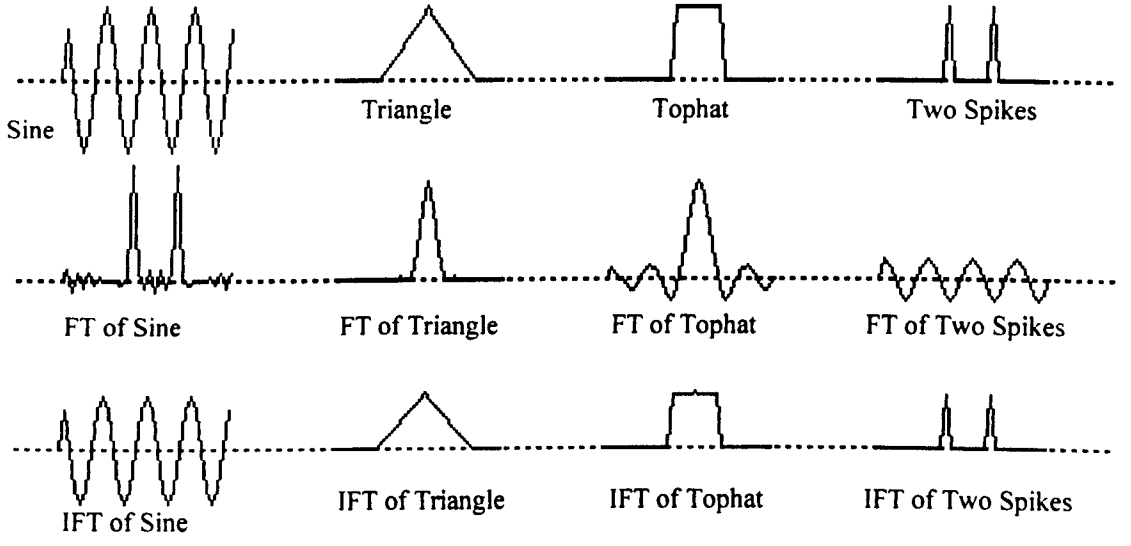


Figure 6.1: Inputs and outputs of tfft1d.cpp to test FT and IFT.

In order to study the spectral behaviour of an analogue signal $f(t)$ in $L^2(0,2\pi)$ from its Fourier transform, full knowledge of the signal in the time-domain must be acquired. This even includes future and past information. In addition, if signal is altered, then the entire spectrum is effected. Indeed, in the extreme case, the Fourier transform of the delta function $\delta(t - t_0)$, with support at single point t_0, is $e^{-it_0\omega}$, which certainly covers the

whole frequency domain. This fact results in taking an infinite amount of computing time. Besides. the formula (6.1.2) which is the inverse Fourier transform, does not even reflects frequencies that evolve with time.

Window function: It is reported in [6] that a function $w \in L^2(\Re)$ is called a **window function** if $tw(t)$ is also in $L^2(\Re)$, where $L^2(\Re)$ denotes the space of square-integrable functions defined on the real line $\Re$. This implies that $\psi \in L^1(\Re)$. For a function $w \in L^2(\Re)$ to qualify as a window function it must satisfy the following conditions:

(i) the function $w(t)$ must decay to zero when $|t|$ tends to infinity; and for all practical purposes the decay should be very fast so that it be a small wave or a *wavelet.*

(ii) It must be possible to identify its center and width.

A window function is called a **wavelet** if it satisfies the condition

$$\int_{-\infty}^{\infty} \psi(x)dx = 0.$$

6.2 Short-time Fourier Transform (Windowed Fourier Transform) STFT

One way to localise the high frequencies while preserving the linearity of the operator is to use windowed Fourier transform also called short-time Fourier transform (STFT). Given a window function $w(t)$ (we require that the function has a finite integral and is non-zero over a finite interval). We define STFT, denoted by $STFT_f(\tau,\omega)$, of a signal $f(t)$ as:

$$STFT_f(\tau,\omega) = \int_{-\infty}^{\infty} f(t)w(t-\tau)e^{-i\omega t}dt \qquad (6.2.1)$$

Observe that the relation is the Fourier transform of the signal with the filter is applied. Moreover, the presence of the filter function $w(t)$ in relation (6.2.1) allows us a window on the frequency spectrum of $f(t)$ around τ. That is, the presence of $w(t-\tau)$ causes the value of $STFT_f(\tau,\omega)$ heavily influenced by the values of $f(t)$ in the neighbourhood of $t=\tau$, with $f(t)$ having essentially no effect, when t is far from τ. This is the significance of the STFT.

The relation (6.2.1) is called the continuous version of the STFT because it defines STFT as a function of continuous time and frequency variables. We need discrete version which is nothing more a continuous version evaluated at the discrete time and frequency points. We usually let these points lie on an equispaced lattice in the time/frequency plane. The spacing in time dimension is taken to be τ_0, and the spacing in the frequency dimension is ω_0. The discrete STFT can therefore also be seen as a function of two integers. One integer, p, specifies the time-dimension coordinate on the lattice. The other, q, specifies the frequency-dimension coordinate[2].

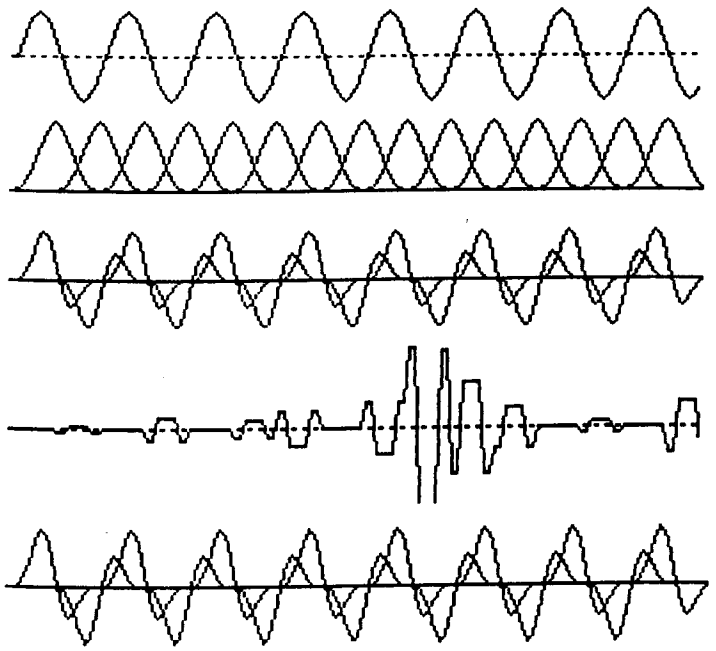
$$STFT_f(p,q) = \int_{-\infty}^{\infty} f(t)w(t - p\tau_0)e^{-iq\omega_0 t}dt\,. \tag{6.2.2}$$

Digitised version (see Section(8.1) for details) of Equation (6.2.2) is given below:

$$STFT_f(p,q) = \sum_{i=0}^{l-1} f(i)w(i+p)e^{-iq\omega_0 i}\,, \text{ where } p = 0,1,....,(l-m) \tag{6.2.3}$$

where ℓ and m denote the signal length and the window width respectively.

Software 6.2.1: A program **tstftp.cpp** (Testing of STFT) is develop to demonstrates the sliding of the window along the input signal depending upon the increment in shift parameter p at fixed but small value of frequency parameter $\omega = qt_0 = 2 * 0.01 = 0.02$. The program is listed in Appendix(1.2) and its outputs are shown in Figure (6.2.1).



(a) The input sine signal.

(b) The Gaussian window sliding along time axis.

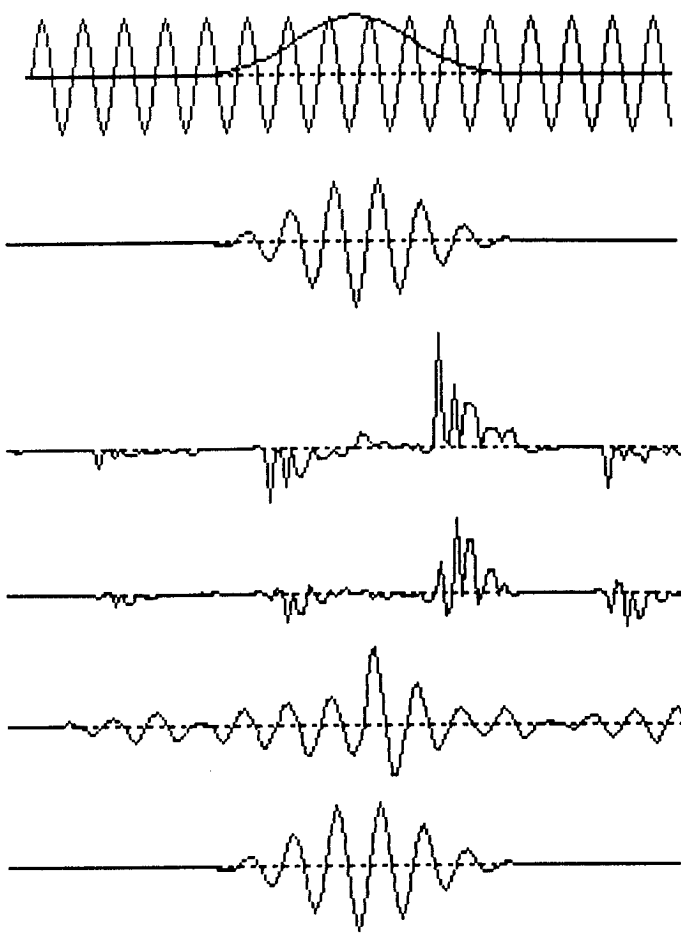
(c) Slices of windowed sine.

(d) STFT of the last slice of sine.

(e) ISTFT of respective windowed slices of sine.

Figure 6.2.1: Outputs of tstftp.cpp to test STFT and sliding of window.

Software 6.2.2: The program named **tstftq.cpp,** is listed in Appendix(1.3). is developed to test and demonstrate the varying effect of frequency on the ISTFT. It can be observed by executing the program that "$\left|\mathrm{Im}(\hat{f}(\omega))\right| \to 0$ as $|\omega| \to 0$, where $\omega = q\omega_0$ is the frequency parameter. If each element of the imaginary part of the spectrum is set equal to zero, the reconstruction of the real time signal is good only for $|\omega| \leq 0.04$. Particularly, for $\omega = 0$. This shows that within this range of frequency, there is no need to compute, store, and transmit the imaginary part of the spectrum; because zero values can be assigned to its elements at the time and place of need in order to reconstruct the time signal". Some of the outputs of the program are shown in Figure (6.2.2).



(a) Sine signal and a Gaussian window.

(b) Windowed sine signal.

(c) Real part of STFT of sine.

(d) Imaginary part of STFT of sine when $\omega = 0.2$.

(e) Reconstructed windowed sine when $\omega = 0.2$.

(f) Reconstructed windowed sine when $|\omega| \leq 0.04$.

Figure 6.2.2: Some outputs of the program tstftq.cpp

6.3 A Serious Redundancy in STFT and its Removal

There are two problems in STFT which are elaborated below:

(i) It is mentioned in Section (6.1) that the spectral contents of the FT spread over the whole frequency domain, even for the delta function $\delta(t - t_0)$ with support at a single point t_0. This drawback can be seen in Figure(6.2.1d) and Figure(6.2.2c).

(ii) Moreover, in Equation(6.2.3) a large number of elements of the windowing function are zero and occur in each window placing. Convolving these zero elements with the time signal is aimless.

Problem(i) is solved automatically when the problem(ii) is solved. The solution of problem(ii) is in sliding the signal in front of a still window. Its advantages and procedure is expressed in Section(8.1). We can apply it for STFT as given bellow:

$$STFT_f(p,q) = \sum_{i=0}^{m-1} f(i+p)w(i)e^{-iq\omega_0 i}, \text{ where } p = 0,1,....,(l-m) \qquad (6.1.3)$$

where ℓ and m denote the signal length and the window width respectively.

Software 6.3: The program **tstftpf.cpp,** is listed in Appendix(1.4) , is to show the removal of computing redundancy from STFT. Some of its results are shown in Figure(6.3).

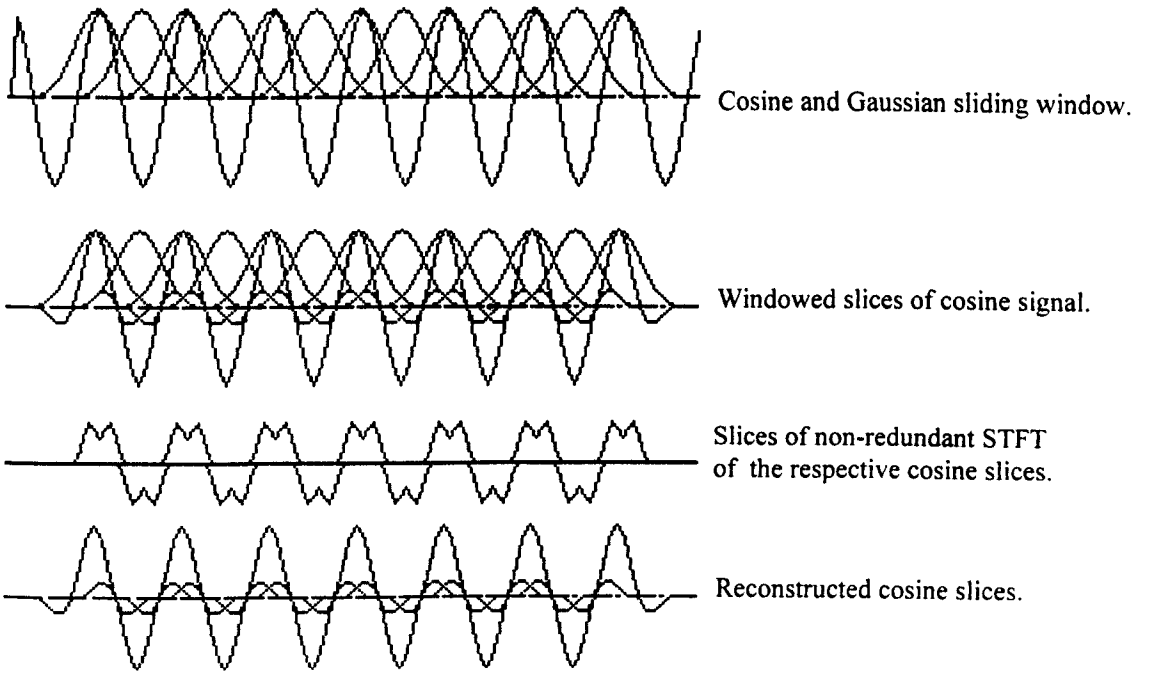


Figure 6.3: Inputs and outputs of tstftpf.exe

6.4 The Gabor Transform

It is stated in [2] that a special case of the STFT in which any of the Gaussian weight function

$$g_\alpha(x) = \frac{e^{\frac{-x^2}{4\alpha}}}{2\sqrt{\pi\alpha}}, \quad \alpha > 0$$

is used as the window function is called the ***Gabor transform***. That is,

$$(G_p^\alpha f)(\omega) = \int_{-\infty}^{\infty} f(t) g_\alpha(t-p) e^{-i\omega_0 qt}\, dt . \tag{6.4.1}$$

Since $\int_{-\infty}^{\infty} e^{-\frac{x^2}{4\alpha}} dx = 2\sqrt{\pi\alpha}$, therefore for $g_\alpha(x) = \frac{e^{\frac{-x^2}{4\alpha}}}{2\sqrt{\pi\alpha}}$, $\alpha > 0$, we have

$$\int_{-\infty}^{\infty} g_\alpha(t-p)dp = \int_{-\infty}^{\infty} g_\alpha(x)dx = 1, \text{ so that } \int_{-\infty}^{\infty} (G_p^\alpha f)(\omega)\, \mathrm{dp} = \hat{f}(\omega).$$

This means that the set $\{ G_p^\alpha f : p \in \Re \}$ of Gabor transform of f decomposes the Fourier transform $\hat{f}$ of f exactly, to give its spectral information.

The Gabor Transform of Spectrum $\hat{f}(\omega)$

It is given in [6] that instead of considering the Gabor transform $(G_p^\alpha f)(\omega)$ given in Equation(6.4.1) as localisation of the Fourier transform of $f(t)$, we may interpret it as windowing the signal $f(t)$ by using the window function $G_{p,\omega}^\alpha(t) = e^{i\omega t} g_\alpha(t-p)$. That is,

$$(G_p^\alpha f)(\omega) = \left\langle f, G_{p,\omega}^\alpha \right\rangle = \int_{-\infty}^{\infty} f(t)\overline{G_{b,\omega}^\alpha(t)}dt . \tag{6.4.2}$$

Then we can relate the Gabor transform of $f(t)$ with the Gabor transform of $\hat{f}(\omega)$ by using the Parseval Identity. That is,

$$(G_p^\alpha f)(\omega) = \left\langle f, G_{p,\omega}^\alpha \right\rangle = \frac{1}{2\pi}\left\langle \hat{f}, \hat{G}_{p,\omega}^\alpha \right\rangle = \frac{e^{-ip\omega}}{2\sqrt{\pi\alpha}} (G_\omega^{1/4\alpha} \hat{f})(-p)$$

This means that, with the exception of the multiple term $\sqrt{\frac{\pi}{\alpha}}e^{-ip\omega}$, the window Fourier transform of $f(t)$ with window function g_α at $t = p$ agrees with the window inverse Fourier transform of $\hat{f}(\omega)$ with window function $g_{1/4\alpha}$ at $\eta = \omega$. The product of the windows is $(2\Delta g_\alpha)(2\Delta g_{1/4\alpha}) = 2$. But if we set

$$H^\alpha_{p,\omega}(\eta) = \frac{1}{2\pi}\hat{G}^\alpha_{p,\omega}(\eta) = \left(\frac{e^{ip\omega}}{2\sqrt{\pi\alpha}}\right)e^{-ip\eta}g_{1/4\alpha}(\eta-\omega),$$

we have $$\left\langle f, G^\alpha_{p,\omega}\right\rangle = \left\langle \hat{f}, H^\alpha_{p,\omega}\right\rangle$$

This means that the information obtained by investigating an analogue signal $f(t)$ at $t = p$ by using the window function $G^\alpha_{p,\omega}(t)$ can also be obtained by observing the spectrum $\hat{f}(\omega)$ of the signal in a neighbourhood of the frequency $\eta = \omega$ by using the window function $H^\alpha_{p,\omega}(\eta)$. Again the product of the window is

$$(2\Delta G^\alpha_{p,\omega})(2\Delta H^\alpha_{p,\omega}) = (2\Delta g_\alpha)(2\Delta g_{1/4\alpha}) = 2$$

6.5 Drawbacks of STFT

It is expressed in [9] that we have acquire the ability to localise the frequencies by having STFT, but we have also acquired some new problems;

1. Inherent to the technique is the fact that we have one more variable.
2. It is not possible to have high accuracy in both the position (in time) and frequency of a contributing discontinuity.
3. If we have a signal consisting of two δ pulses in time, they can not be discriminated by STFT using this $w(t)$ if they are Δt apart or less. Similarly two pure sine waves (δ pulses in frequency) can not be discriminated if they are Δf apart or less. We can improve the frequency discrimination by choosing a $w(t)$ with smaller Δf, and similarly for time, but unfortunately they are not independent. In fact there is an equality, the Heisenberg inequality that bounds their product:

$$\Delta t\ \Delta f \geq 1/4\pi$$

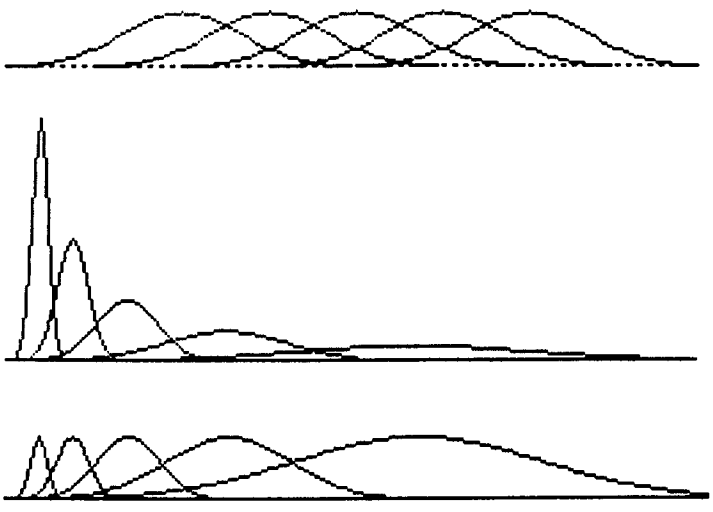
4. Once the window is chosen, the resolution limit is the same over all times and frequencies. This means that there is no adaptability of the analysis, and if we want good resolution of the short bursts, we have to sacrifice good frequency description of the long smooth sections.

6.6 Relative Frequency Analysis

What is really needed for one to be able to determine the time intervals that yield the spectral information on any desirable range of frequencies ?. In addition, since the frequency of a signal is directly proportional to the length of its cycle, it follows that for higher-frequency spectral information, the time-interval should be relatively small to give better accuracy, and for low-frequency spectral information, the time interval should be relatively wide to give complete information. In other words, it is important to have a flexible time-frequency window that automatically narrows at high center-frequencies and widen at low center-frequencies. Fortunately, the integral wavelet transform has this so-called *zoom-in and zoom-out* capability and will be main issue in next chapters.

It is also stated in [9] that one obvious fix to this problem is to let the window Δt, and therefore the Δf vary as a function of the frequency. A simple relation is to require $\Delta f / f = c$, where c is a constant. The aim is to increase the resolution in time for sharp discontinuities while keeping a good frequency resolution at high frequencies. Of course if the signal is composed of high frequencies of long duration (as is a very noisy signal), this strategy does not pay off, but if the signal is composed of relatively long smooth areas separated by well-localised sharp discontinuities (as in many real or computer-generated images and scenes) then this approach will be effective.

Software 6.6.1: A program named **trband.cpp** is listed in Appendix(1.5) and its outputs is shown in Figure(6.6.1). The fact that "a system of constant relative bandwidth windows is achieved from a relative bandwidth windows by multiplying all windows with a constant so that their heights be the same" can be seen in the listing of the program.



(a) A constant bandwidth window.

(b) A relative bandwidth window.

(c) A constant relative bandwidth window (herec=2).

Figure 6.6.1: A constant, a relative, and a constant relative bandwidth windows.

6.7 Switching from STFT to Continuous Wavelet Transform

It is stated in [9] that we can choose any set of windows to achieve the constant relative bandwidth, but a simple version is if all the windows are scaled version of each other. To simplify notation, let us define $\psi(t)$ as:

$$\psi(t) = w(t)e^{-i2\pi\omega_0 t}$$

and scaled version of $\psi(t)$:

$$\psi_a(t) = \frac{1}{\sqrt{|a|}}\psi(\frac{t}{a})$$

where a is the scale factor (that is $\omega = \frac{\omega_0}{a}$), and the constant $\frac{1}{\sqrt{|a|}}$ is for energy normalisation. The STFT now becomes:

$$STFT(\tau,a) = \int_{-\infty}^{\infty} f(t)\psi(\frac{t-\tau}{a})dt\,.$$

This is known as ***Wavelet Transform*** and $\psi(t)$ is called the ***basic wavelet.*** Changing notation, we have Wavelet Transform (WT) and Inverse Wavelet Transform (IWT) given as:

$$WT(\tau,a) = \int_{-\infty}^{\infty} f(t)\psi(\frac{t-\tau}{a})dt \qquad (6.8.1)$$

$$f(t) = c\int_{-\infty}^{\infty} WT(\tau,a)\psi(\frac{t-\tau}{a})dt \qquad (6.8.2)$$

It is clear from the above formula that the basic wavelet is scaled, translated, and convolved with the signal to compute the transform. The translation corresponds to moving the window over the time signal, and the scaling, which is often called dilation in the context of wavelet, corresponds to the filter frequency bandwidth scaling.

6.8 Software for Wavelet Transform

Software 6.8.1: To demonstrate WT and IWT with relative frequency bandwidth, the program **trbwt11.cpp** is listed in Appendix(1.6) and its outputs are shown in Figure(6.8.1).

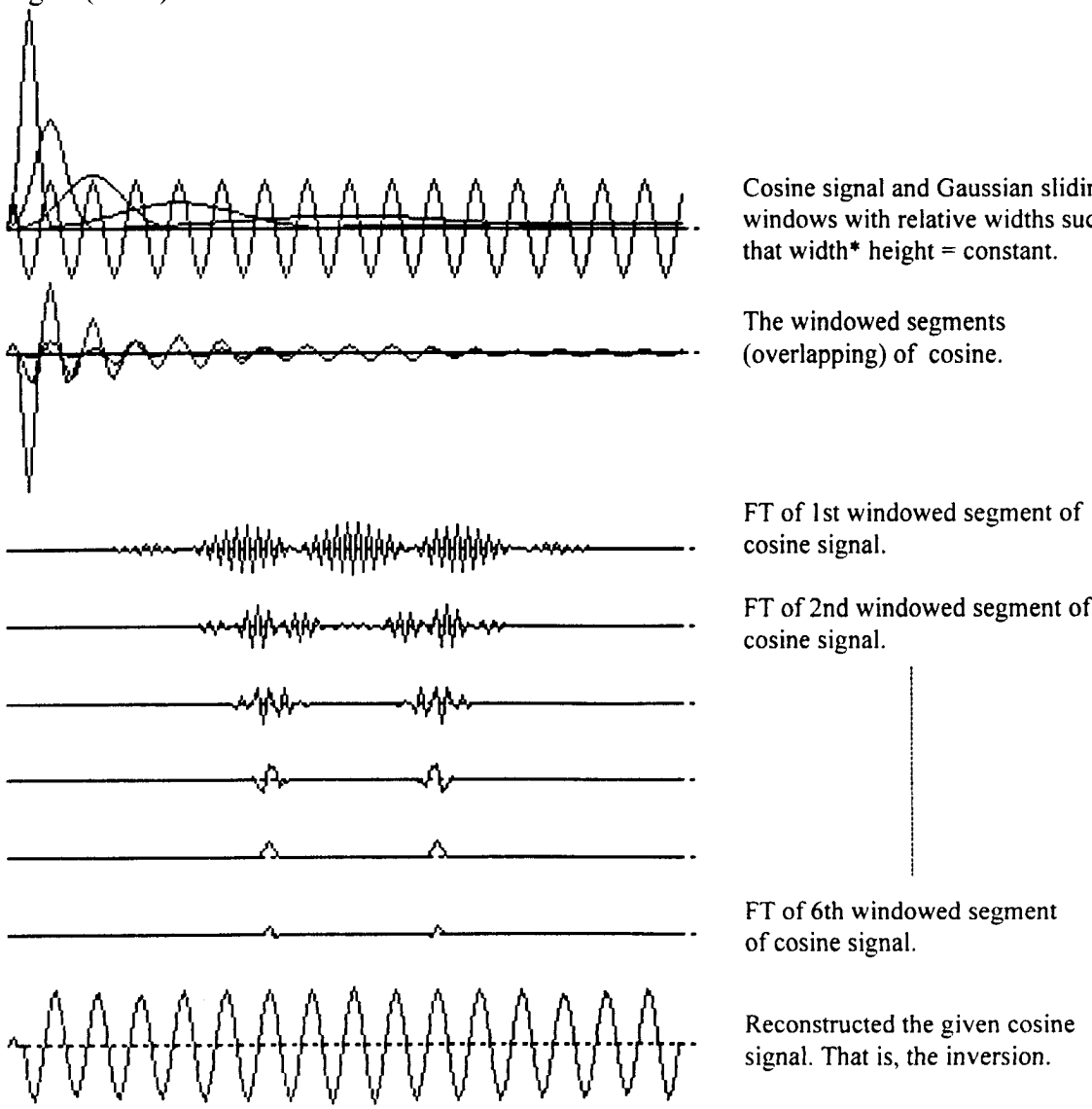


Figure 6.8.1: Outputs of trbwt11.cpp to demonstrate relative bandwidth Wavelet Transform and Inverse Wavelet Transform.

Software 6.8.2: To demonstrate WT and IWT with constant relative bandwidth, a program named **tcrbwt11.cpp** is listed in Appendix(1.7) and some of its outputs are shown in Figure(6.8.2).

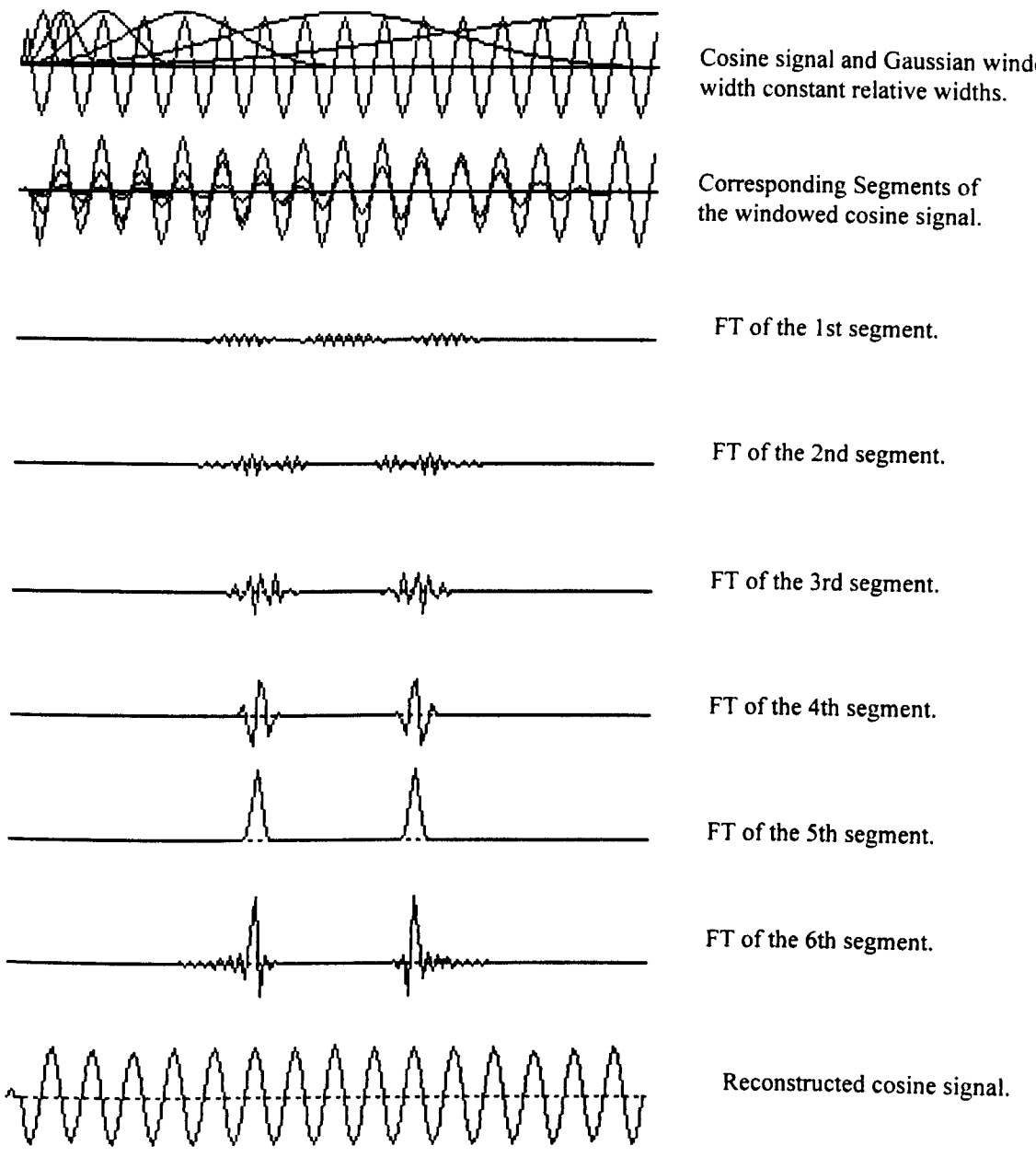


Figure 6.8.2: Outputs of tcrbwt11.cpp to demonstrate WT and its inverse with constant relative bandwidth.

Software 6.8.3: To demonstrate multiresolution WT with relative bandwidth. a program **tmrwt11.cpp** is listed in Appendix(1.8) and some of its outputs are shown in Figure(6.8.3).

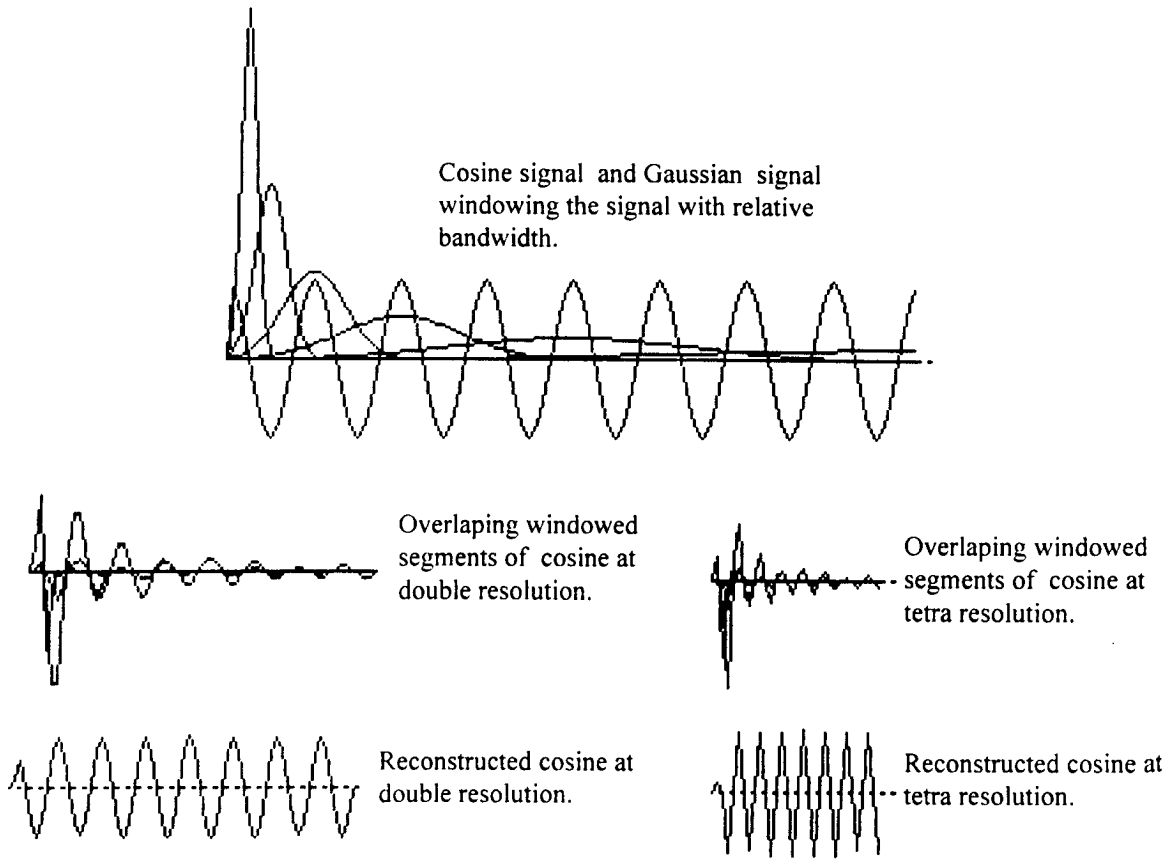


Figure 6.8.3: Some output of tmrwt11.cpp for multiresolution relative bandwidth WT and IWT.

6.9 Decomposition and Reconstruction Algorithms

We know that both $f_j \in V_j$ and $g_j \in W_j$ have unique representations[6]:

$$f_j(x) = \sum_l c_l^j \phi(2^j x - l), \quad \text{with } \mathbf{c}^j = \left\{c_l^j\right\} \in l^2(\Re);$$

$$g_j(x) = \sum_l d_l^j \psi(2^j x - l), \quad \text{with } \mathbf{d}^j = \left\{d_l^j\right\} \in l^2(\Re);$$

Decomposition:

$$\begin{aligned} f_{j-1}(x) + g_{j-1}(x) &= f_j(x) \\ &= \sum_l c_l^j \phi(2^j - l) \\ &= \sum_l c_l^j \left[\sum_k \left\{ p_{l-2k} \phi(2^{j-1} - l) + q_{l-2k} \psi(2^{j-1} - l) \right\} \right] \end{aligned}$$

$$= \sum_k \left\{ \sum_l p_{l-2k} c_l^j \right\} \varphi(2^{j-1}x - k) + \sum_k \left\{ \sum_l q_{l-2k} c_l^j \right\} \psi(2^{j-1}x - k)$$

$$\Rightarrow \sum_k \left\{ \sum_l p_{l-2k} c_l^j \right\} \varphi(2^{j-1}x - k) - f_{j-1}(x) + \sum_k \left\{ \sum_l q_{l-2k} c_l^j \right\} \psi(2^{j-1}x - k) - g_{j-1}(x) = 0$$

$$\Rightarrow \sum_k \left\{ \sum_l p_{l-2k} c_\ell^j - c_l^{j-1} \right\} \varphi(2^{j-1}x - k) + \sum_k \left\{ \sum_l q_{l-2k} c_l^j - d_l^{j-1} \right\} \psi(2^{j-1}x - k) = 0$$

ℓ^2 - linear independence of $\{\phi_{j-1,k} : k \in Z\}$ and $\{\psi_{j-1,k} : k \in Z\}$ and the fact that $V_{j-1} \cap W_{j-1} = \{0\}$ yields

$$\begin{cases} c_k^{j-1} = \sum_l p_{l-2k} c_l^j \\ d_k^{j-1} = \sum_l q_{l-2k} c_l^j \end{cases}$$

If we use non negative indices for the coefficients p_i's and q_i's then the above relation can equivalently be written as

$$\begin{cases} c_k^{j-1} = \sum_{l-2k} p_{l-2k} c_l^j \\ d_k^{j-1} = \sum_{l-2k} q_{l-2k} c_l^j \end{cases} \tag{6.9.1}$$

Here the sequences $\mathbf{c}^j = \{c_l^j,\ l \in Z\}$ and $\mathbf{d}^j = \{d_l^j,\ l \in Z\}$ represents the digitized form of $f_j(x)$ and $g_j(x)$ respectively[6].

Reconstruction:

For $\mathbf{c}^j = \{c_k^j\} \in l^2(\Re)$, we know that

$$\sum_k c_k^j \phi(2^j x - k) = f_j(x)$$

$$= f_{j-1}(x) + g_{j-1}(x)$$

$$= \sum_l c_l^{j-1} \phi(2^{j-1}x - l) + \sum_l d_l^{j-1} \psi(2^{j-1}x - l)$$

$$= \sum_l \left[c_l^{j-1} \sum_k p_k \phi(2^{j-1}x - 2l - k) + d_l^{j-1} \sum_k q_k \phi(2^{j-1}x - 2l - k) \right]$$

$$= \sum_l \sum_k (c_l^{j-1} p_{k-2l} + d_l^{j-1} q_{k-2l}) \phi(2^j x - k)$$

$$= \sum_k \left\{ \sum_l p_{k-2l} c_l^{j-1} + q_{k-2l} d_l^{j-1} \right\} \phi(2^j x - k)$$

Hence l^2 - linear independence of $\{\phi_{j-1,k} : k \in Z\}$ implies that

$$c_l^j = \sum_l \left[p_{k-2l} c_l^{j-1} + q_{k-2l} d_l^{j-1} \right]$$

If we use non negative indices for the coefficients a_l's and b_l's then the above relation can equivalently be written as

$$c_l^j = \sum_{k-2l} \left[p_{k-2l} c_l^{j-1} + q_{k-2l} d_l^{j-1} \right]$$

Here the sequence $\mathbf{c}^j = \left\{ c_l^j,\ l \in Z \right\}$ represent the digitized form of $f_j(x)$ [6].

6.10 An Example Algorithm for Wavelet Decomposition and Reconstruction

Decomposition:

Consider D4 scaling function $\{\phi(x)\}$ and D4 wavelet function $\{\psi(x)\}$ given by

$$\{\phi(x)\} = \left\{ \sqrt{2}(1+\sqrt{3})/4,\ \sqrt{2}(3+\sqrt{3})/4,\ \sqrt{2}(3-\sqrt{3})/4,\ \sqrt{2}(1-\sqrt{3})/4 \right\}$$

$$\{\psi(x)\} = \left\{ (1-\sqrt{3})/4\sqrt{2},\ -(3-\sqrt{3})/4\sqrt{2},\ (3+\sqrt{3})/4\sqrt{2},\ -(1+\sqrt{3})/4\sqrt{2} \right\}$$

We can use both the function $\{\phi(x)\}$ and $\{\psi(x)\}$ as the coefficients $\{p(l-2k)\}$ and $\{q(l-2k)\}$, respectively, in the relation (6.9.1). So that

$$\begin{cases} c^{j-1}(k) = \sum_{l-2k} \phi(l-2k) c^j(l) \\ d^{j-1}(k) = \sum_{l-2k} \psi(l-2k) c^j(l) \end{cases}$$

By replacing $\mathbf{c}^j = \left\{ c_l^j,\ l \in Z \right\}$ and $\mathbf{d}^j = \left\{ d_l^j,\ l \in Z \right\}$ by the actual functions $f_j(x)$ and $g_j(x)$, we have

$$\begin{cases} f^{j-1}(k) = \sum_{l-2k} \phi(l-2k) f^j(l) \\ g^{j-1}(k) = \sum_{l-2k} \psi(l-2k) f^j(l) \end{cases} \qquad (6.10.1)$$

For a given function f_j, we want to compute the functions g_{j-1}, g_{j-2},..., g_{j-m} so that the given function f_j can be written as

$$f_j = g_{j-1} + g_{j-2} + \ldots + g_{j-m} + f_{j-m} \qquad (6.10.2)$$

If F, Φ, and Ψ denote the signal f_j, the scaling function $\{\phi(x)\}$, and the wavelet function $\{\psi(x)\}$ respectively, then Relations (6.10.1) can be written in the matrix form as

$$\begin{cases} F^{j-1} = \Phi F^{j} \\ G^{j-1} = \Psi F^{j} \end{cases}, \text{ where } F = [f_0\ f_1\ \ldots\ f_{n-1}]', \text{ and}$$

$$\Psi = \begin{bmatrix} \psi(0) & \psi(1) & \psi(2) & \psi(3) & 0 & 0 & 0 & \ldots & & & & & & & 0 \\ 0 & 0 & \psi(0) & \psi(1) & \psi(2) & \psi(3) & 0 & \ldots & & & & & & & 0 \\ \ldots & & & & & & & & & & & & & & \\ \ldots & & & & & & & & & & & & & & \\ \ldots & & & & & & & & & & & & & & \\ \ldots & & & & & & & & & & & \psi(0) & \psi(1) & \psi(2) & \psi(3) \\ \psi(2) & \psi(3) & 0 & 0 & \ldots & & & & & & & 0 & 0 & \psi(2) & \psi(3) \end{bmatrix}$$

The matrix Φ can be written in the same shape.

Reconstruction: Since $\{\psi_{0,k}(x)\}$ is such that $\Psi'\Psi = I$, therefore $\Psi^{-1} = \Psi'$. Similarly, $\Phi^{-1} = \Phi'$. If we denote the function $g_{j-1}(x)$ by the matrix G^{j-1}, then

$$F^{j} = \Psi' G^{j-1} + \Phi' F^{j-1}$$

Repeated application of this relation yields

$$F^{j} = \Psi' G^{j-1} + \Psi' G^{j-2} + \ldots + \Psi' G^{j-m} + \Phi' F^{j-m}$$

6.11 A Wavelet Decomposition leading to a Wavelet Transform

Objective: In this section we will develop a wavelet decomposition which will be upgraded in next section to have a wavelet transform.

System of windows:

1. We wish to have minimum number of windows in terms of which a real signal can be decomposed. The Gaussian windows are found the best.
2. A Gaussian window of particular width can be used to have **constant bandwidth wavelet decomposition.**
3. For **relative bandwidth wavelet decomposition,** a suitable Gaussian function can be used to generate a system of relative bandwidth wavelets.
4. The system of relative bandwidth wavelets can slightly be modified to have a **system of constant relative bandwidth wavelets (SCRBW)** in which the heights of the wavelets are equal while the widths are constant multiples of the basic wavelet.

Mathematical Description:

A function $f(x)$ can be decomposed according to Equation(6.10.2) written as

$$f(x) = \sum_{n} g_n(x) f(x) + err(x) \qquad (6.11.1)$$

where the functions $g_n(x)$ consists of a system of constant, or relative, or constant relative bandwidth Gaussian windows shown in Figure(6.6). Important features of the system are given below:

(1) The functions $g_n(x)$ are defined by

$$g_j(x) = \frac{1}{2\sqrt{\pi 2^j}} \exp(-(x - cj)^2 / 2^{j+2}), \quad j = 1,2,...6 \qquad (6.11.2)$$

where c is an integre in [0,j] and window *width* $= 2^j$ $(= 2\sqrt{\alpha}$, $\alpha = 2^{2j-2}$) .

(2) Equation(6.11.2) provides a relative bandwidth. But we are interested in constant relative bandwidth which can be achieved by multiplying each window with its width.

(3) Further, we multiply each window with 2 in order to have normalized windows.

(4) The shift parameter $k = cj$, say, is helpful to minimise the error function $err(x)$.

(5) For $j > 6$, the function given in Equation(6.11.2) results in non localised windows; hence are of no use.

(6) The resemblance of the shape of $err(x)$ function with the input signal $f(x)$ depends upon the choice of the system of windows. If the resemblance is in phase with $f(x)$, then although its existence is additive even then it is of scaling nature and can therefore be dropped to have

$$f(x) \approx \sum_{n} g_n(x) f(x) \qquad (6.11.3)$$

(7) Equation(6.11.3) provides us a good approximation of the input signal $f(x)$. However, the result can be improved in a number of ways when required.

Improving the Approximation: Approximation can be improved by adopting any of the following suggestions:

(i) A seventh window can be included in the system by copying the localised part of the seventh Gaussian window shown in Figure(6.11.1) at its proper index position and declaring the non-localised elements of the window equals zero.

Software 6.11.1: To demonstrate the formation of this window, a program **tw7gauss.cpp** is listed in Appendix(1.9).

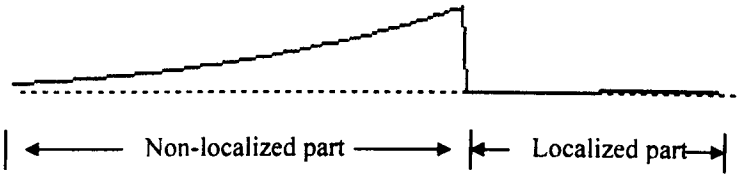


Figure 6.11.1: The seventh Gaussian window which is partially localized.

(ii) A seventh window can be constructed by copying the first part of sixth window as its second half while first be declared zero.

(iii) A suitable linear combination of the first halves of the fifth and sixth window can also be used as the second half of the seventh window as expressed in (ii).

It will become clear in nest section that the above suggested improvement is good for wavelet decomposition but it is not necessary for wavelet transform to be achieved there.

Software 6.11.2: To demonstrate the constant bandwidth wavelet decomposition of a signal by using a number of copies of single Gaussian window, a program named **twdcband.cpp** is listed in Appendix(1.10) and its outputs are shown in Figure(6.11.2).

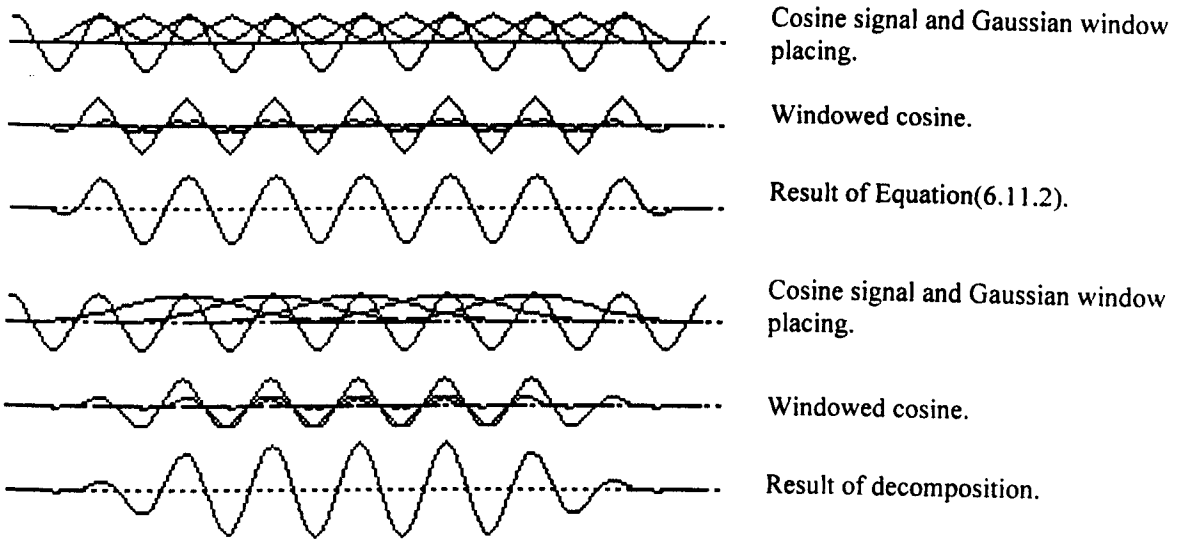
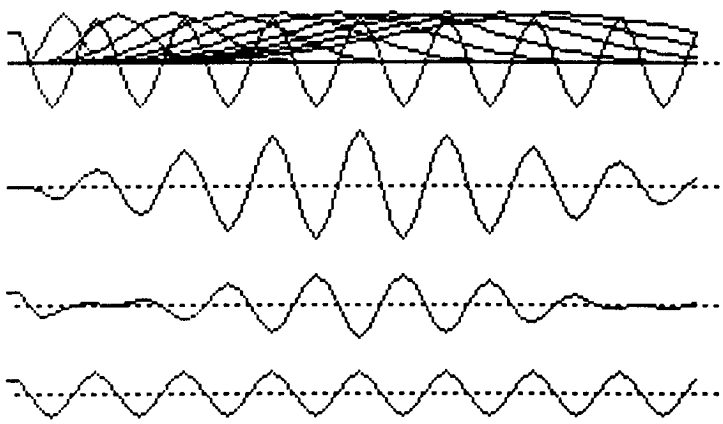


Figure 6.11.2: Outputs of twdcband.cpp to demonstrate constant bandwidth wavelet decomposition.

Software 6.11.3: To demonstrate SCRBW given in Equation(6.11.1) by using 10 Gaussian windows given in Equations(6.11.2) with width equals $2j$ and shift parameter equals j. a program named **twdcomp1.cpp** is listed in Appendix(1.11) and its outputs are shown in Figure(6.11.3).



(a) Cosine signal and Gaussian window placing.

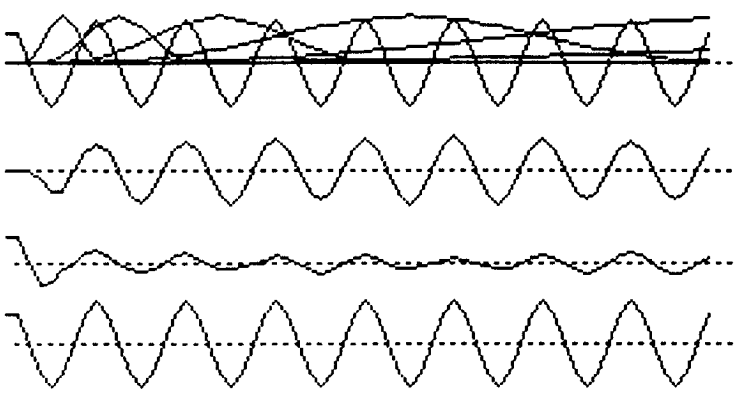
(b) Approximation of cosine; Result of Equations(6.11.2).

© The error function which is not of the scaling nature in this case; since it resembles with cosine in negative sense.

(d) The sum of the signals given in (b) and (c) above.

Figure 6.11.3 : Outputs of twdcomp1.cpp to demonstrate SCRBW.

Software 6.11.4: To demonstrate SCRBW given in Equation(6.11.1) by using 6 Gaussian windows given in Equations(6.11.2) when the size of input signal and hence of Gaussian function is 64, a program named **twdcomp2.cpp** is listed in Appendix(1.12) and its outputs are shown in Figure(6.11.4).



(a) Cosine signal and the window placing.

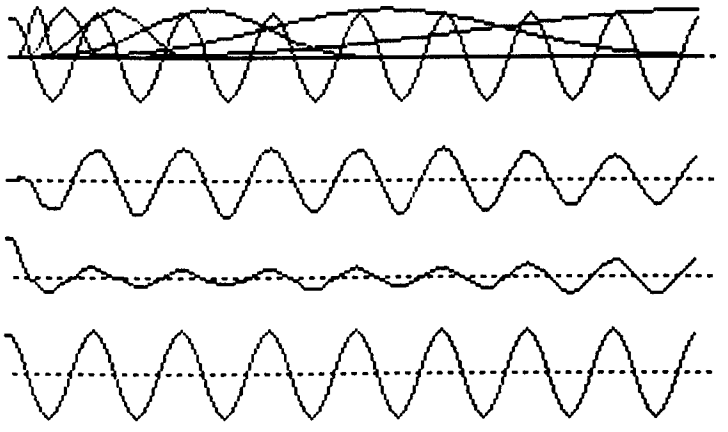
(b) Approximation of cosine. Result of Equations(6.11.2).

© The error function which is of of scaling nature in this case;

(d) The sum of the signals given in (b) and © above.

Figure 6.11.4: Outputs of twdcomp2.cpp to demonstrate SCRBW.

Software 6.11.5: To demonstrate SCRBW given in Equation(6.11.1) by using 6 Gaussian windows given in Equations(6.11.2) when the size of input signal and hence of Gaussian function is 128, a program named **twdcomp3.cpp** is listed in Appendix(1.13) and its outputs are shown in Figure(6.11.5). Notice that the approximation given in Figure(6.11.5(b)) is less refined as compared to that in Figure(6.11.4(b)).



(a) Cosine signal and the window placing.

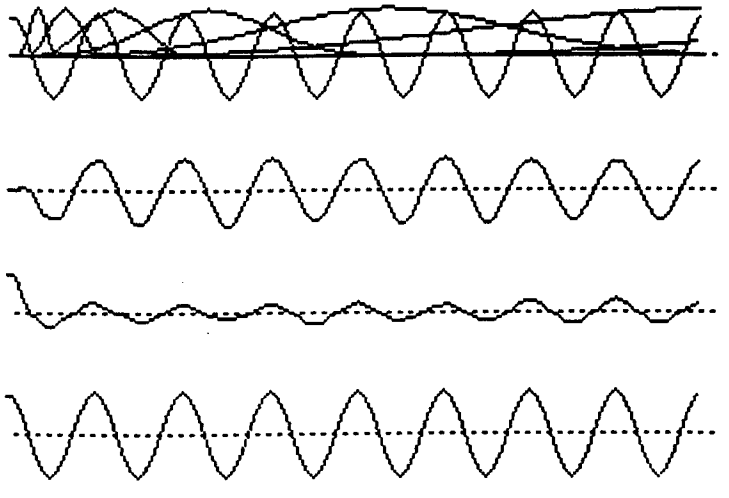
(b) Approximation of cosine which is less refined as compared to that in Figure(6.11.6).

© The error function which is of of scaling nature but goes higher at the end.

(b) The sum of the signals given in (b) and © above.

Figure 6.11.5: Outputs of twdcomp3.cpp to demonstrate SCRBW.

Software 6.11.6: To demonstrate the improvement of SCRBW given in Software(6.11.4) by including the seventh window suggested in Section(6.11.3), a program named **twdcomp4.cpp** is listed in Appendix(1.14) and its outputs are shown in Figure(6.11.6).



(a) Cosine signal and the placing of 7 windows.

(b) Approximation of cosine which is refined as compared to that in Figure(6.11.5) due to 7th window.

© The error function which is good to be dropped as compared to that in Figure(6.11.5©).

(b) The sum of the signals given in (b) and © above.

Figure 6.11.6: Outputs of twdcomp4.cpp to demonstrate SCRBW.

Software 6.11.7a: To demonstrate the multiresolution wavelet decomposition by using SCRBW. a program named **twdmrrb.cpp** is listed in Appendix(1.15) and its outputs are shown in Figure(6.11.7).

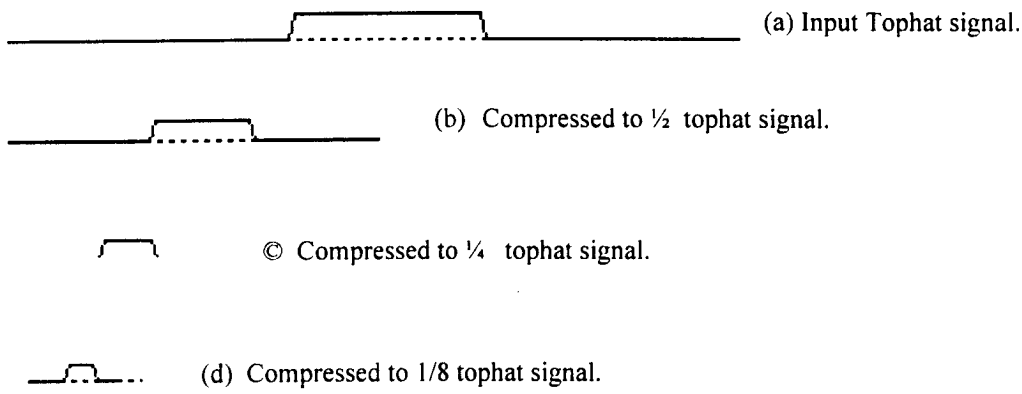


Figure 6.11.7: Outputs of twdmrrb.cpp to demonstrate multiresolution wavelet decomposition.

Software 6.11.7b: To demonstrate the multiresolution wavelet decomposition by using SCRBW, the same program named **twdmrrb.cpp** which used is in section Software (6.11.7a) and is listed in Appendix(1.15). Here input signal is cosine instead of the tophat.

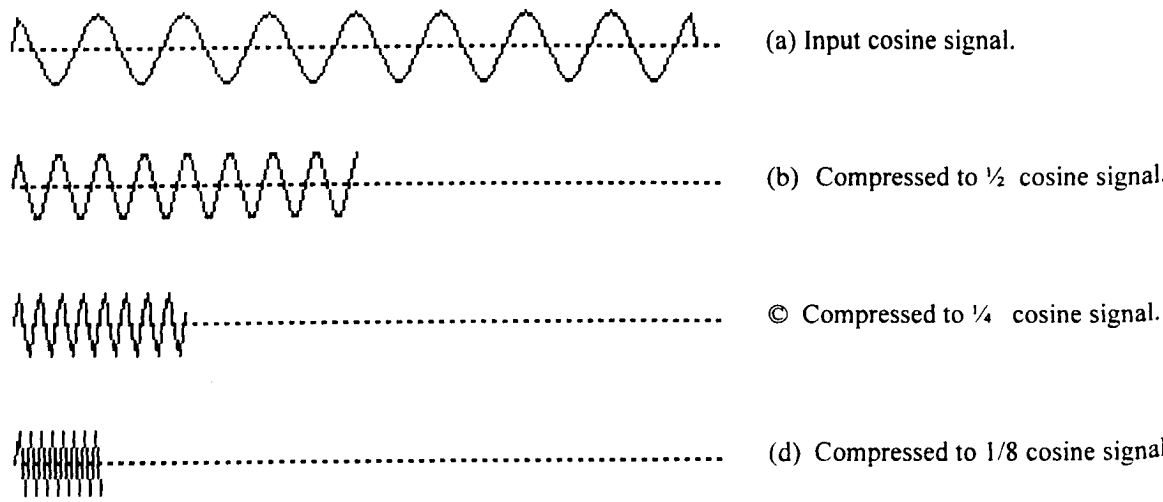


Figure 6.11.7b: Outputs of twdmrrb.cpp to demonstrate multiresolution wavelet decomposition.

6.12 Associating two Wavelet Transforms with the Wavelet Decomposition

In this section we will associate two wavelet transforms with the wavelet decomposition presented in the previous section. Let us write the system of Gaussian wavelets given in Equations(6.11.2), after multiplying with the constant factor 2 and relative normalising factor *width* equals 2^k, as

$$g_k(x) = \frac{2^k}{\sqrt{\pi 2^k}} \exp(-(x-ck)^2 / 2^{k+2}), \quad k = 1,2,...6 \qquad (6.12.1)$$

where c is an integre in [0,k] and window *width* $= 2^k$ $(= 2\sqrt{\alpha}\ ,\ \alpha = 2^{2k-2}\ )$.

We can write the given system $\{g_k(i)\}_{i=1}^n$, $(k = 1,2,....,6)$ of wavelets in the matrix form, denoted by $\mathbf{G}$, as

$$\mathbf{G} = \begin{bmatrix} g_1(1) & g_1(2) & \ldots\ldots & g_1(n) \\ g_2(1) & g_2(2) & \ldots\ldots & g_2(n) \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ g_6(1) & g_6(2) & \ldots\ldots & g_6(n) \end{bmatrix}$$

Then the function $\{\psi_i(k)\}_{i=1}^6$ defined by

$$\{\psi_i(k)\}_{i=1}^6 = \frac{1}{\lambda_k}\{g_k(i)\}_{i=1}^n \ , \quad k = 1,2,....,6.$$

represents the normalised *kth* column of $\mathbf{G}$, where λ_k denotes the energy of the column. If $f(x)$ is a real signal to be transformed (decomposed), then for each fixed k we have

$$\overset{w}{f}_k = \sum_{i=1}^{6} \psi_i(k) f(k) \qquad (6.12.2)$$

$$\sum_{i'}^{6} \psi_{i'}(k) \overset{w}{f}_k = \sum_{i'}^{6} \psi_{i'}(k) \left\{ \sum_{i=1}^{6} \psi_i(k) f(k) \right\} = \sum_{i,i'=1}^{6} f(k) \psi_i(k) \psi_{i'}(k) = \sum_{i,i'=1}^{6} f(k) \delta_{i,i'}$$

For $i' = i$, we have

$$\sum_{i=1}^{6} \psi_i(k) \overset{w}{f}_k = f(k) \qquad (6.12.3)$$

Equation(6.12.2) and Equation(6.12.3) are called **Discrete Wavelet Transform** (DWT) and the **Discrete Inverse Wavelet Transform** (DIWT) respectively. Where the set $\{\psi_i(k)\}_{i=1}^6$

forms the **orthonormal basis**. Since the basis have achieved from the wavelets therefore should be called **Wavelet Basis**. Moreover, since the wavelets are decomposing, therefore, the basis should be called **Wavelet Basis** associated with the **Wavelet Decomposition** (WBWD).

It should be noted that for practical purposes, we can compute DWT and DIWT without normalising the columns of $\mathbf{G}$ and hence without involving the set $\{\psi_i(k)\}_{i=1}^{6}$, as given below:

$$\overset{w}{f}_k = \sum_{i=1}^{6} g_k(i) f(k) \quad \text{and} \quad f(k) = \frac{\overset{w}{f}_k}{c_k}, \quad \text{where } c_k = \sum_{i=1}^{6} g_i(k), \quad k = 1,2,....n \tag{6.12.4}$$

The basis WBWD are orthonormal since (i) they are normalised and (ii) each element $f(k)$ of the function $f(x)$ is uniquely expressed by the *kth* column $\{\psi_i(k)\}_{i=1}^{6}$. Uniqueness is implied by the fact that if there is some other set $\{\varphi_i(k)\}_{i=1}^{6}$ in terms of which the element $f(k)$ can be expressed then the set $\{\varphi_i(k)\}_{i=1}^{6}$ do not constitute the *kth* column of the matrix $\mathbf{G}$ such that the rows of the matrix can produce the same Gaussian wavelets. More clearly, each element $g_k(i)$ of the matrix $\mathbf{G}$ is treated as it were the only non-zero element of a matrix $\mathbf{A}_{ik}$ of order 6x*n* so that $\mathbf{G} = \sum_{i=1}^{6}\sum_{k=1}^{n} \mathbf{A}_{ik}$.

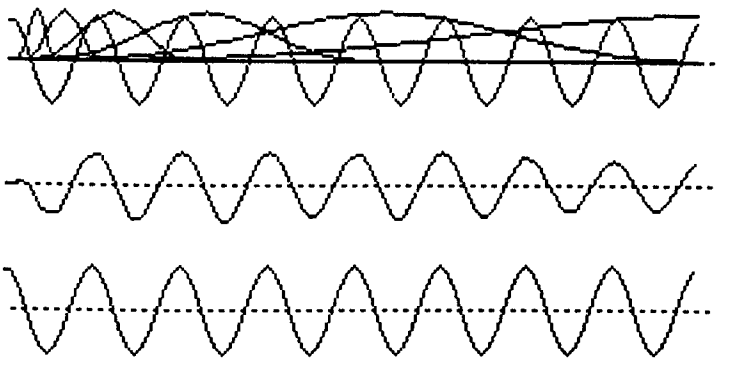
There is an other important wavelet transform, along the rows of the matrix $\mathbf{G}$, which should also be associated with the wavelet decomposition. We will express it along with its application to noise reduction and data compression in Chapters(7-9) in detail, however, we can define it immediately as:

$$f_j(k) = \frac{2^{j/2}}{\lambda} \sum_{i=1}^{l} f(i-1)\psi(2^j i - k), \text{ where } k = 0,1,....,2^j(l-m) \tag{6.12.5}$$

where $[0,l[$ denotes the domain of both the input function f and the wavelet function ψ and m ($m < l$) be the number of non-zero elements of ψ that constitute a window.

Software 6.12: There are three software programs developed to test and demonstrate the DWT and IDWT given in Equation(6.12.3). The programs and their details follow:

Software 6.12.1: To demonstrate DWT and DIWT, a program named **twt&it1.cpp** is listed in Appendix(1.16) and its outputs are shown in Figure(6.12.1).



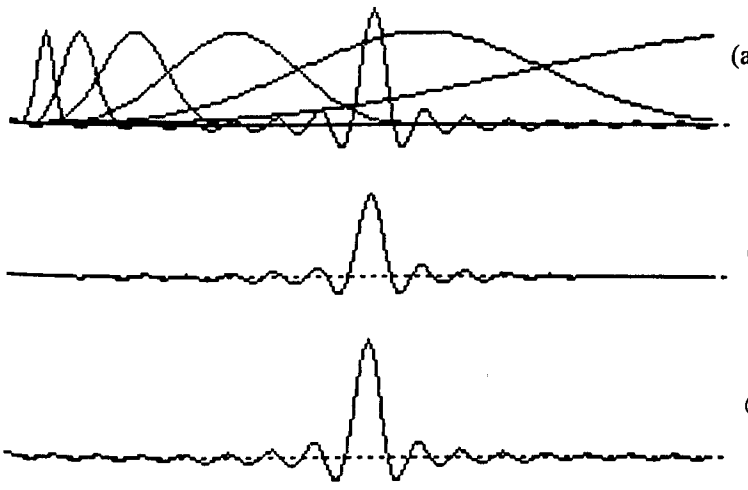
(a) The Window placing and the input cosine signal.

(b) DWT/wavelet decomposition of the input signal.

© DIWT. That is, reconstruction of cosine.

Figure 6.12.1: Outputs of twt&it2.cpp to demonstrate DWT and DIWT.

Software 6.12.2: To demonstrate DWT and DIWT, a program named **twt&it2.cpp** is listed in Appendix(1.17) and its outputs are shown in Figure(6.12.2).



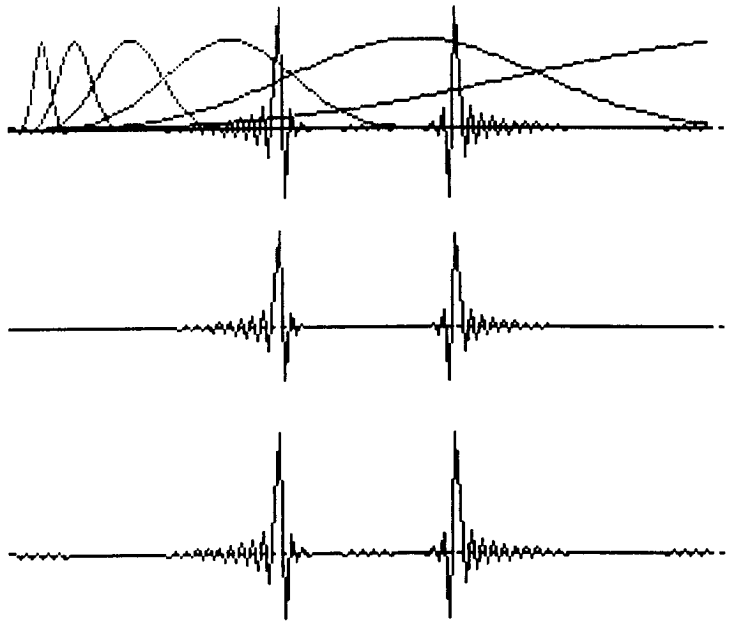
(a) The placing of Gaussian windows and the input signal which is the FFT of Tophat function.

(b) DWT/ wavelet decomposition of the given input signal.

© Inverse of DWT. That is, reconstructed input signal.

Figure 6.12.2: Outputs of twt&it1.cpp to demonstrate DWT and DIWT.

Software 6.12.3: To demonstrate DWT and DIWT, a program named **twt&it3.cpp** is listed in Appendix(1.18) and its outputs are shown in Figure(6.12.3).



(a) Windows placing and the input signal which is FFT of a windowed cosine.

(b) DWT/wavelet decomposition of the given input signal.

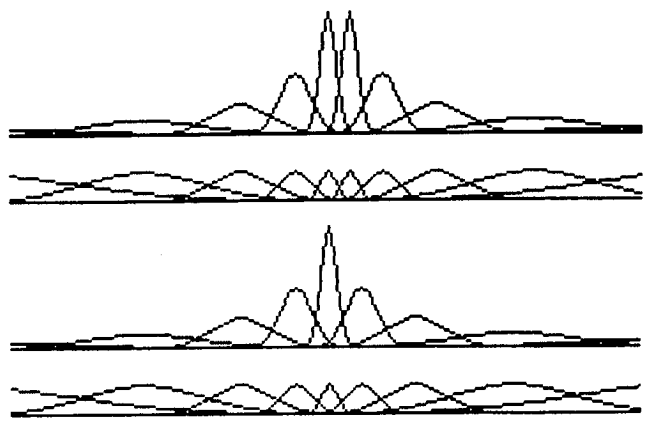
© DIWT. That is, the reconstructed input signal.

Figure 6.12.3: Output of twt&it3.cpp to demonstrate DWT and DIWT.

6.13 Developing Some Wave Packets

Some systems of constant relative bandwidth wavelets, may be called wavelet packets, can be developed which are applicable for noise reduction and data compression of input signals having both the high and the low frequency contents. Different wavelet packets are tested in following subsections.

Software 6.13.1: To demonstrate the formation of central high frequency wavelet packets, a program named **wpacket1.cpp** is listed in Appendix(1.19) and its outputs are shown in Figure(6.13.1).



(a) A relative bandwidth wavelet packet with two central high frequency peaks.

(b) Constant relative wavelet packet constructed from that in (a).

© A relative bandwidth wavelet packet with one central high frequency peaks.

(d) Constant relative wavelet packet constructed from that in (c).

Figure 6.13.1: Outputs of wpacket1.cpp to demonstrate the formation of wavelet packets.

Software 6.13.2: To demonstrate the formation of a central low frequency wavelet packet, a program named **wpacket2.cpp** is listed in Appendix(1.20) and its outputs are shown in Figure(6.13.2).

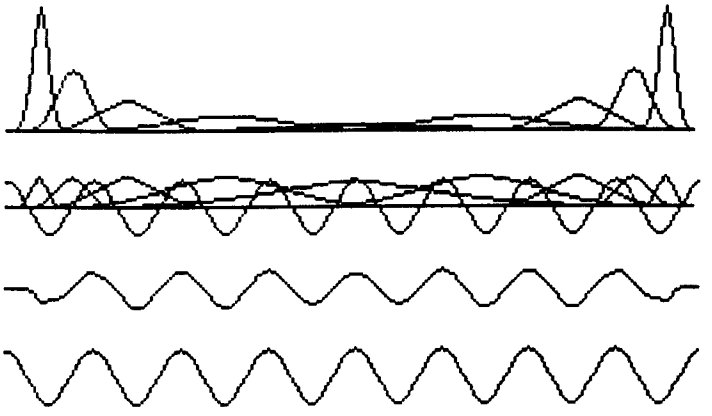


Figure 6.13.2: Outputs of wpacket2.cpp to demonstrate the formation and application of a low central frequency wavelet packet.

Software 6.13.3: To demonstrate the formation of a central low frequency wavelet packet, a program named **wpacket3.cpp** is listed in Appendix(1.21) and its outputs are shown in Figure(6.13.3).

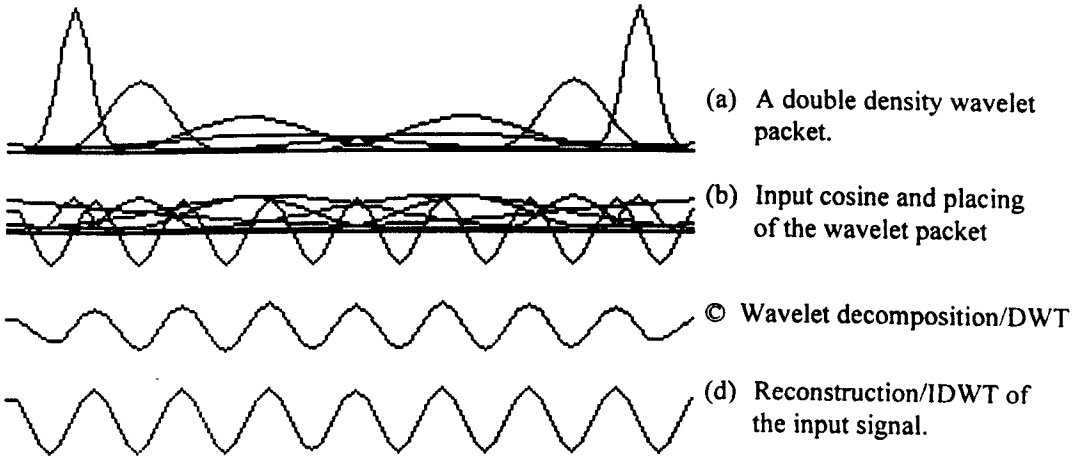


Figure 6.13.3: Outputs of wpacket3.cpp to demonstrate the formation and application of a double density wavelet packet.

6.14 A Binary Wavelet Packet as Orthonormal Basis and its Continuous WT

This Section is divided into three subsections that follow.

6.14.1 A system of Constant Relative Bandwidth Binary Coded (CRBBC) Gaussian Windows

Consider the system of constant relative bandwidth six Gaussian windows defined in Equations(6.12.1) given by

$$g_k(x) = \frac{2^k}{\sqrt{\pi 2^k}} \exp(-(x-ck)^2 / 2^{k+2}), \quad k = 1.2,...6$$

where c is an integre in [0,k] and window $width = 2^k$ $(= 2\sqrt{\alpha}\ ,\ \alpha = 2^{2k-2}\ )$.

We achieve binary coding, denoted by $\{g_k^b(i)\}_{i=1}^n$, of the system $\{g_k(i)\}_{i=1}^n$ by applying a three stage hard transforming function f_h, as given below:

$$g_k^b(i) = f_h(g_k(i)) \tag{6.14.1}$$

If $\max g$ denotes the maximum valued element of the system $\{g_k(i)\}_{i=1}^n$, f_h is defined as

$$f_h(x) = \begin{cases} -1, & \text{whent} \quad 0 \le x < \dfrac{\max g}{3} \\ 0, & \text{whent} \quad \dfrac{\max g}{3} \le x < \dfrac{2\max g}{3} \\ 1, & \text{whent} \quad \dfrac{2\max g}{3} \le x < \max g \end{cases} \tag{6.14.2}$$

The CRBBC system given in Equations(6.14.1) is important both for the wavelet transform and for pattern recognition. These issues will be discussed in Section(6.14.2) and Section(6.14.3) respectively.

Software 6.14.1: A software program **tcrbbcg.cpp** (Testing the formulation of Constant Relative Bandwidth Binary Coded Gaussian windows) is developed and listed in Appendix(1.22). The outputs of the program are shown in Figure(6.14.1).

(a) The system of six Gaussian windows.

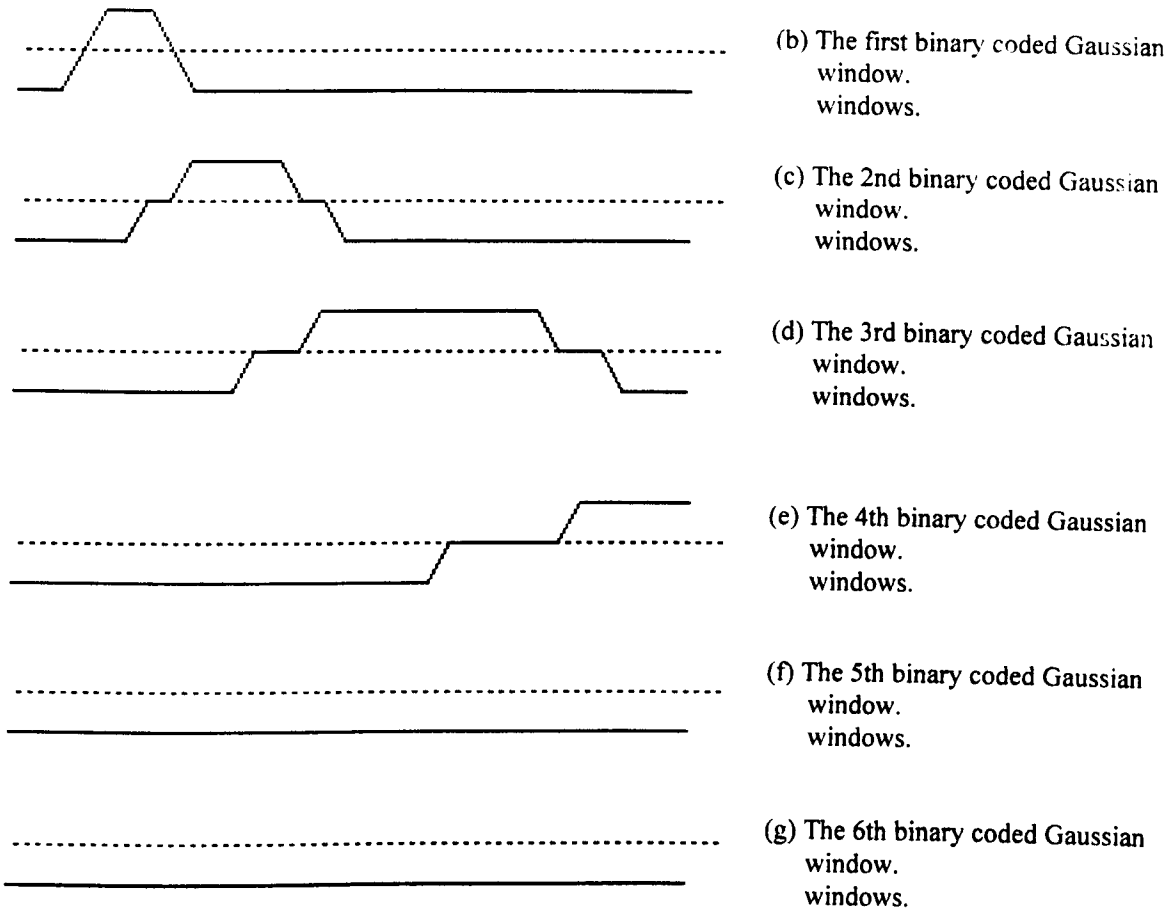


Figure 6.14.1: Outputs of tcrbbcg.cpp to test and demonstrate the formulation constant relative bandwidth binary coded Gaussian windows.

6.14.2 A Binary Wavelet Basis Transform

In contrast with Section(6.12), We have here the binary coding of the system $\{g_k(i)\}_{i=1}^{n}$ in the form $\{g_k^b(i)\}_{i=1}^{n}$; we therefore write the binary coded matrix $\mathbf{G}_b$, for $n=16$, as given below:

$$\mathbf{G}_b = \begin{bmatrix} g_1^b(1)\ g_1^b(2) \ldots\ldots g_1^b(16) \\ g_2^b(1)\ g_2^b(2) \ldots\ldots g_2^b(16) \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ g_6^b(1)\ g_6^b(2) \ldots\ldots g_6^b(16) \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 & 0 & 1 & 1 & 1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Then the binary coding of the functions $\{\psi_i(k)\}_{i=1}^{6}$ defined as

$$\left\{\psi_i^b(k)\right\}_{i=1}^{6} = \frac{1}{\lambda_k^b}\left\{g_k^b(i)\right\}_{i=1}^{n} \quad . \quad k = 1,2.....6.$$

represents, like earlier, the normalised *kth* column of $\mathbf{G}_b$, where λ_k^b denotes the energy of the column.

Similarly, by replacing $\left\{\psi_i(k)\right\}_{i=1}^{6}$ with $\left\{\psi_i^b(k)\right\}_{i=1}^{6}$ from Equations(6.12.2) and Equations(6.12.3), we have the required wavelet transform of a real signal $f(x)$ and its inverse transform respectively as

$$\overset{w}{f}_k = \sum_{i=1}^{6} \psi_i^b(k) f(k) \tag{6.14.3}$$

$$f(k) = \sum_{i=1}^{6} \overset{w}{f}_k \psi_i^b(k) \tag{6.14.4}$$

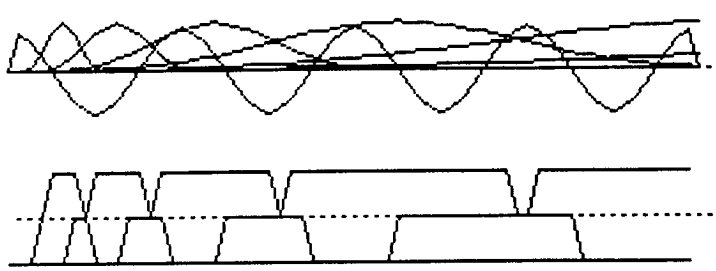
where the set $\left\{\psi_i^b(k)\right\}_{i=1}^{6}$ constitute the orthonormal binary basis associated with the wavelet packet, and hence the transform can be termed as a **Binary Wavelet Basis Transform.**

Like earlier, for practical purposes we compute the transform and the inverse transform as

$$\overset{w}{f}_k = \sum_{i=1}^{6} f(k) g_k^b(i) \quad \text{and} \quad f(k) = \frac{\overset{w}{f}_k}{c_k}, \quad \text{where } c_k = \sum_{i=1}^{6} g_i^b(k), \quad k = 1,2,....n. \tag{6.14.5}$$

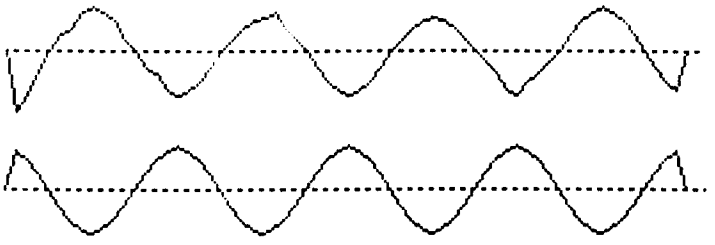
The other wavelet transform, applicable to noise reduction and data compression, which is associated along the rows of the matrix $\mathbf{G}_b$ is maintained and will be expressed in Section(6.14.3).

Software 6.14.2: A software program **twtcrbcg.cpp** (Testing of a Wavelet Transform using Constant Relative(bandwidth) Binary Coded Gaussian(windows)) is developed and listed in Appendix(1.23). The outputs of the program are shown in Figure(6.14.2).



(a) Input cosine signal and placing of the Gaussian windows.

(b) The system of binary coded Gaussian windows. windows.



© The transformed output signal using the binary coded Gaussian windows.

(d) The reconstructed/IWT signal.

Figure 6.14.2: Outputs of twtcrbcg.cpp to test and demonstrate wavelet transform using constant relative bandwidth binary coded Gaussian windows.

6.14.3: A Binary Wavelet Packet Transform

As mentioned earlier, the second wavelet transform associated with the wavelet decomposition is the noise reducing plus data compressing Continuous Wavelet Transform. Although a wavelet transform using the family of a single basic wavelet, either binary coded or not, does work well, but to get some additional advantages we develop an unusual wavelet transform that uses six families of the binary coded wavelet packet given in Equation(6.14.1).

For a single wavelet g^b the digitised version of continuous wavelet is defined as (see details in Chapter7-8) given below:

$$f_j(k) = \frac{2^{j/2}}{\lambda}\sum_{i=1}^{l} f(i-1)g^b(2^j i - k), \text{ where } k = 0,1,....,2^j(l-m) \qquad (6.14.6)$$

where the input real function f is defined on $[0,l[$, and m $(m < l)$ is the width of the wavelet g^b.

If $\{g_r^b(i)\}_{i=1}^{n}$, (r = 1,2,...,6) is the wavelet packet defined in Equation(6.14.1), then we define a transform as

$$f_j(k) = \frac{2^{j/2}}{\lambda}\sum_{r=1}^{6}\sum_{i=1}^{l} f(i-1)g_r^b(2^j i - k), \quad k = 0,1,....,2^j(l - wwp) \qquad (6.14.7)$$

where wwp denotes width of the wavelet packet. This transform can be termed as **Binary Wavelet Packet Transform**. We prefer feed-forward connections and multi-choice switches in developing an ANN. The multi-choice switches are desired to avoid feed back connections as well. This transform leads to such an ANN. There are two switches inherited; one is associated with the parameter r, $1 \le r \le 6$ which gives the number of

wavelets to be activated simultaneously - higher the value of r the more noise will be reduced; and the other is associated with the value of wwp, $2^2 \le wwp \le 2^7$ such that the smaller values more suits for higher-frequency contents and vice versa. It should be noted that both of the switches can either be operated manually or preferably can work automatically in co-ordination with some noise/frequency detector circuits as suggested in Section(10.2).

The binary nature of the wavelet packet is advantageous for its hardware implementation and storage. The connections will obviously be simple and the operations of binary bits requires only the low level basic logical gates instead of general purpose micro-processors. The complete packet can be stored as only six binary numbers with maximum suggested length of 128 sign bits each. The storage can be visualised as a matrix-like memory array of 6x128 sign bits which can easily be partitioned vertically into 6 by octaves for the selection of the value of wwp, and from this partition any number of rows can be chosen occasionally as the value of r.

Software 6.14.3: A software program **tmrbcgwt.cpp** (Testing of the Binary Wavelet Packet Transform) is developed and listed in Appendix(1.24). The outputs of the program are shown in Figure(6.14.3). It is appropriate to point out here that the wavelet packet is used as a single wavelet. This also demonstrate that a two-dimensional wavelet can be applied even to one-dimensional signals.

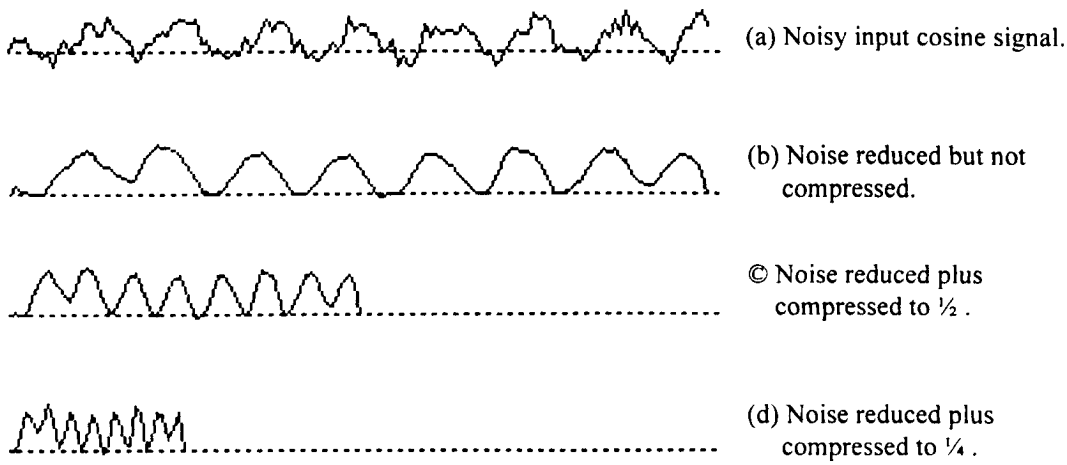


Figure 6.14.3: Outputs of tmrbcgwt.cpp to demonstrate performance of binary wavelet packet transform.

6.15 A Special Development for Relative and Constant Relative Bandwidth

Suppose that the function $\phi(x) = 2^n\phi(2^n x - k)$ is a wavelet. Then it can easily be shown that

$$\phi(x) = \sum_k c_k \phi(2^n x - k) \Leftrightarrow \int_0^\infty \phi(x)dx = \int_0^\infty \phi(2^n x - k)d(2^n x - k) = 1, \text{ and } \sum_k c_k = 2^n$$

Let the function $\phi(x)$ having support $[0,k]$ be defined as

$$\varphi(x) = \frac{1}{m}\begin{cases} 1, & 0 \le x \le m, \text{ where } m \text{ is an integer and } m \ge k \in Z \\ 0, & \text{otherwise} \end{cases}$$

then we can solve the refinement equation

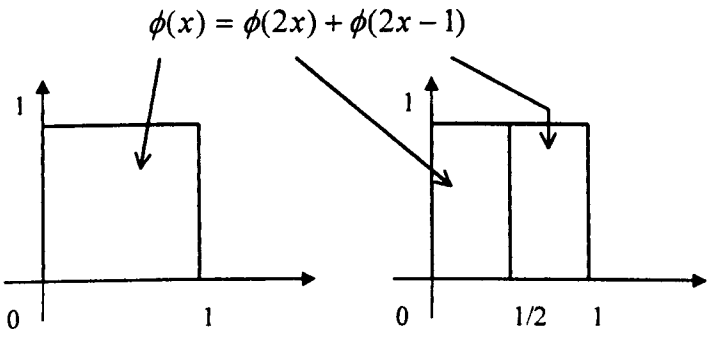
$$\phi(x) = \sum_k c_k \phi(2x - k), \quad \sum_k c_k = 2$$

when $c_0 = 1$, $c_m = 1$, m can have a value from the set $]0,...,1/3,1/2,1,2,3,...,k]$; and all other c_k's are zero.

Case-1: When $m = 1$, then $\phi(x) = \phi(2x) + \phi(2x - 1)$ gives the decomposition of the box function which can be described as follows:-

$$\varphi(2x) = \begin{cases} 1, & 0 \le 2x \le 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 1, & 0 \le x \le 1/2 \\ 0, & \text{otherwise} \end{cases}$$

$$\varphi(2x - 1) = \begin{cases} 1, & 0 \le 2x - 1 \le 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 1, & 1/2 \le x \le 1 \\ 0, & \text{otherwise} \end{cases}$$

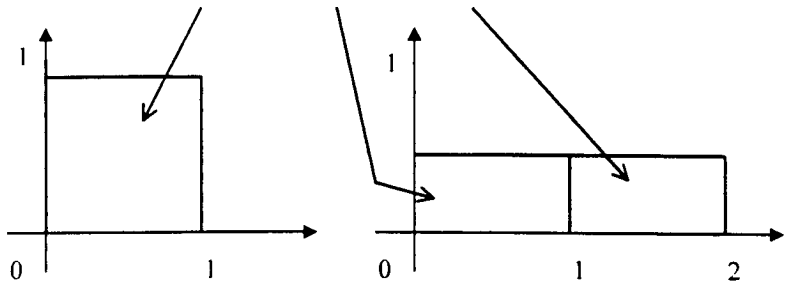


Case-2: When $m = 2$, $\phi(x) = \phi(2x) + \phi(2x - 2)$ gives the rectangle of unit area with $length = 2$, $width = 1/2$ and given by

$$\varphi(2x) = \frac{1}{2}\begin{cases} 1, & 0 \le 2x \le 2 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 1/2, & 0 \le x \le 1 \\ 0, & \text{otherwise} \end{cases}$$

$$\varphi(2x-2) = \frac{1}{2}\begin{cases} 1, & 0 \le 2x-2 \le 2 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 1/2, & 1 \le x \le 2 \\ 0, & \text{otherwise} \end{cases}$$

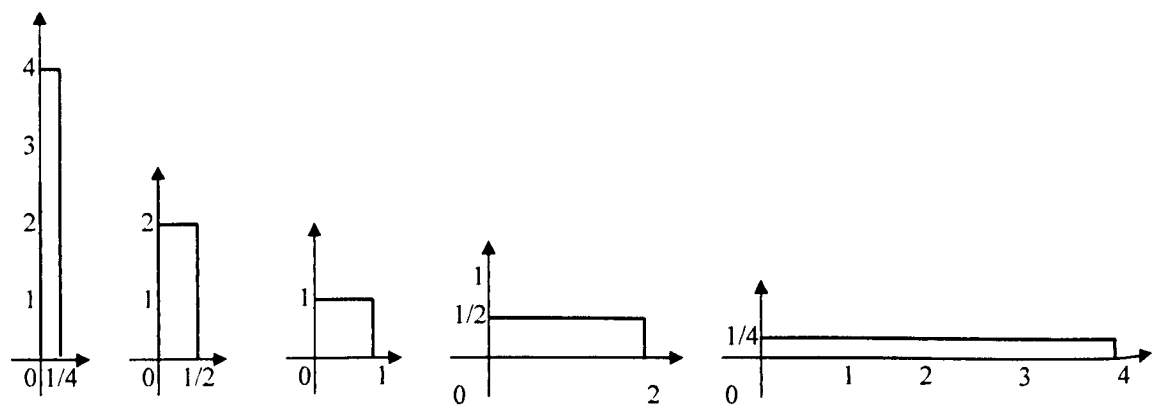
$$\phi(x) = \phi(2x) + \phi(2x-2)$$



In general, $\phi(x) = \phi(2x) + \phi(2x-m)$ gives a rectangle of unit area with $length = m$, $width = 1/m$, where

$$\varphi(2x) = \frac{1}{m}\begin{cases} 1, & 0 \le 2x \le m \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 1/m, & 0 \le x \le m/2 \\ 0, & \text{otherwise} \end{cases}$$

$$\varphi(2x-m) = \frac{1}{m}\begin{cases} 1, & 0 \le 2x-m \le m \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 1/m, & m/2 \le x \le m \\ 0, & \text{otherwise} \end{cases}$$



Where

$\phi_2(x)$, $\phi_1(x)$, $\phi_0(x)$, $\phi_{-1}(x)$, and $\phi_{-2}(x)$ corresponds to $m = 1/4, 1/2, 1, 2, 4$ respectively. It is to be noted that the above development can, equivalently and more sophisticatedly, be achieved in a different way which is given in the following. The procedure adopted gives us the general multiresolution iterative formulas.

Alternate Solution:

For j=0,1,2,-1,-2: we plot $\phi_j(x) = 2^{j-1}\sum_k c_k \phi_j(2^j x - k)$, I=[$2^{-j}$k,$2^{-j}$(k+1)]; where $\sum_k c_k = 2$, such that c_0=2, and all other c_k are zero; and $\phi_0(x) = \begin{cases} 1, & 0 \le x < 1 \\ 0, & \text{otherwise} \end{cases}$

For j=0,1,2; we have

$$\phi_0(x) = \begin{cases} 1, & 0 \le x < 1 \\ 0, & \text{otherwise} \end{cases}$$

$$\phi_1(x) = 2\phi_0(2x) = 2\begin{cases} 1, & 0 \le 2x < 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 2, & 0 \le x < 1/2 \\ 0, & \text{otherwise} \end{cases}$$

$$\phi_2(x) = 2\phi_1(2x)$$

$$= 2^2\phi_0(2^2 x)$$

$$= 4\phi_0(4x) = 4\begin{cases} 1, & 0 \le 4x < 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 4, & 0 \le x < 1/4 \\ 0, & \text{otherwise} \end{cases}$$

In general, we have

$\phi_j(x) = 2^{j-1}\phi(2^j x)$ and so $\lim_{j\to\infty}\phi_j(x) = \delta$, where δ is the delta function

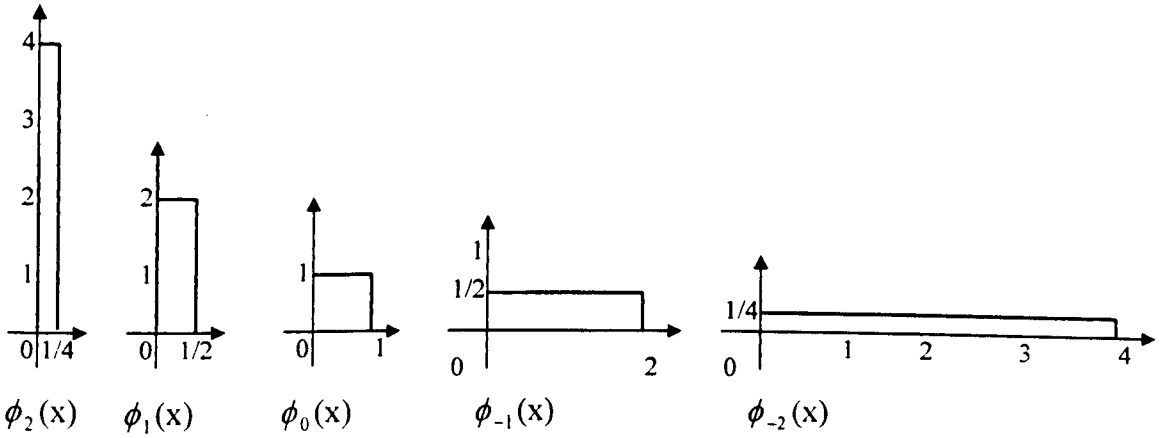
For j=-1,-2, we have

$$\phi_{-1}(x) = \frac{1}{2}\phi_0(x/2) = \frac{1}{2}\begin{cases} 1, & 0 \le x/2 < 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 1/2, & 0 \le x < 2 \\ 0, & \text{otherwise} \end{cases}$$

$$\phi_{-2}(x) = \frac{1}{2}\phi_{-1}(x/2)$$

$$= \frac{1}{4}\phi_0(x/4) = 1/4\begin{cases} 1, & 0 \le x/4 < 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} 1/4, & 0 \le x < 4 \\ 0, & \text{otherwise} \end{cases}$$

In general, we have $\phi_{-j}(x) = 2^{-j}\phi(2^{-j}x)$ and so $\lim_{j\to\infty}\phi_{-j}(x) = 0$. The graphs of the functions $\phi_2(x)$, $\phi_1(x)$, $\phi_0(x)$, $\phi_{-1}(x)$, and $\phi_{-2}(x)$ are shown in the following;



$\phi_2(x)$ $\phi_1(x)$ $\phi_0(x)$ $\phi_{-1}(x)$ $\phi_{-2}(x)$

6.16 Multiresolution in High pass Filtering

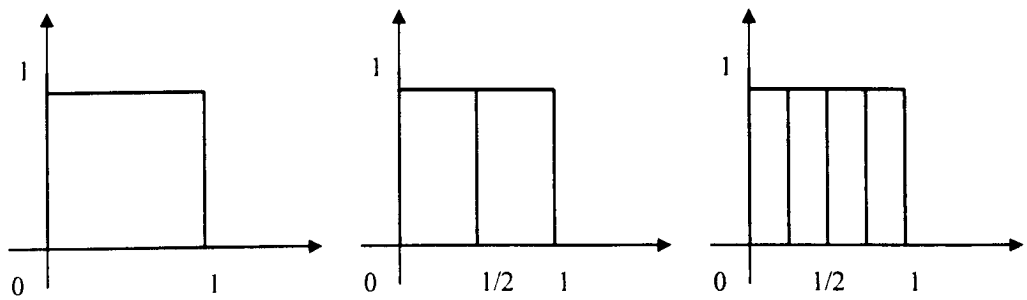
The fact that the *high pass filtering* a signal keeps its scale, but increases its resolution visualised in the following cases of refinement equation.

Case-1: For n=1,2; Plot $\phi^n(x) = \sum_k c_k \phi^{n-1}(2x-k)$, I=[$2^{-1}$k,$2^{-1}$(k+1)];

where $\sum_k c_k = 2$, such that $c_0 = c_1 = 1$, and all other c_k's are zero; and

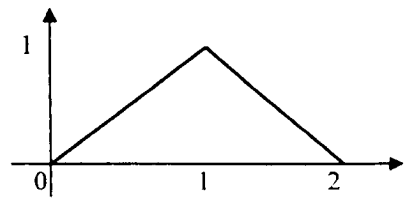
$$\phi(x) = \begin{cases} 1, & 0 \le x < 1 \\ 0, & \text{otherwise} \end{cases}$$

Solution: Graphs of the functions, $\phi(x)$, $\phi^1(x) = \phi^0(2x) + \phi^0(2x-1)$, and $\phi^2(x) = \phi^1(2x) + \phi^1(2x-1) = \phi^0(4x) + \phi^0(4x-1) + \phi^0(4x-2) + \phi^0(4x-3)$ are shown below:



Case-2: For j = 0,1; Plot $\phi_j(x) = \sum_k c_k \phi_{j-1}(2^j x - k)$, where $\sum_k c_k = 2$, such that $c_1=1$, $c_0=c_2=1/2$, and all other c_k's are zero; and $\phi_0(x) = \begin{cases} x, & 0 \le x < 1 \\ 2-x, & 1 \le x < 2 \end{cases}$

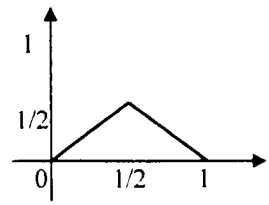
Solution: For j=0, we have $\phi_0(x) = \begin{cases} x, & 0 \le x < 1 \\ 2-x, & 1 \le x < 2 \end{cases}$



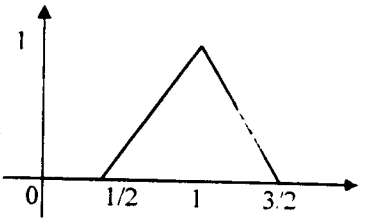
For j=1, we can write $\phi_1(x) = \frac{1}{2}\phi_0(2x) + \phi_0(2x-1) + \frac{1}{2}\phi_0(2x-2)$.

Where

$$\frac{1}{2}\phi(2x) = \frac{1}{2}\begin{cases} 2x, & 0 \le 2x < 1 \\ 2-2x, & 1 \le 2x < 2 \end{cases} = \begin{cases} x, & 0 \le x < 1/2 \\ 1-x, & 1/2 \le x < 1 \end{cases}$$

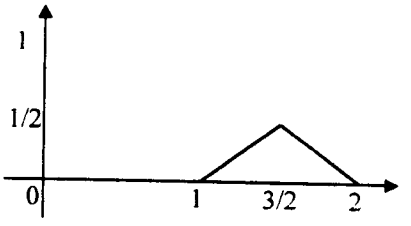


$$\phi(2x-1)=\begin{cases}2x-1, & 0\le 2x-1<1\\ 2-(2x-1), & 1\le 2x-1<2\end{cases}=\begin{cases}2x-1, & 1/2\le x<1\\ 3-2x, & 1\le x<3/2\end{cases}$$



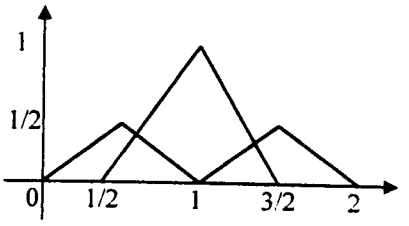
$$\frac{1}{2}\phi(2x-2)=\frac{1}{2}\begin{cases}2x-2, & 0\le 2x-2<1\\ 2-(2x-2), & 1\le 2x-2<2\end{cases}$$

$$=\begin{cases}x-1, & 0\le x<3/2\\ 2-x, & 3/2\le x<2\end{cases}$$



Combining the three components, we have

$$\phi(x)=\frac{1}{2}\phi(2x)+\phi(2x-1)+\frac{1}{2}\phi(2x-2)$$



It can be seen that $\phi(x)$ is symmetric about $x=1$ and its integral is 1.

$$\int_0^2\varphi(x)dx=\frac{1}{2}\int_0^1\varphi(2x)dx+\int_{1/2}^{3/2}\varphi(2x-1)dx+\frac{1}{2}\int_1^2\varphi(2x-2)dx$$

$$=\int_0^{1/2}xdx+\int_{1/2}^1(1-x)dx+\int_{1/2}^1(2x-1)dx+\int_{1/2}^{3/2}(3-2x)dx+\int_1^{3/2}xdx+\int_{3/2}^2(1-x)dx$$

$$=\int_0^{1/2}xdx+\int_{1/2}^1(1-x)dx+\int_{1/2}^1(2x-1)dx+\int_1^{3/2}(3-2x)dx+\int_1^{3/2}xdx+\int_{3/2}^2(1-x)dx$$

$$=\frac{x^2}{2}\Big|_0^{1/2}+(x-\frac{x^2}{2})\Big|_{1/2}^1+(x^2-x)\Big|_{1/2}^1+(3x-x^2)\Big|_1^{3/2}+\frac{x^2}{2}\Big|_1^{3/2}+(x-\frac{x^2}{2})\Big|_{3/2}^2$$

$$=\frac{1}{4}+\frac{1}{2}+\frac{1}{4}=1$$

The sequence $\left\{\frac{1}{4},\frac{1}{2},\frac{1}{4}\right\}$ is important and it will be used as a wavelet basis.

Case-3: For j=1,2; Plot $\phi_j(x)=\sum_k c_k\phi_{j-1}\left(2^jx-k\right)$, I=[$2^{-1}$k,$2^{-1}$(k+1)];

where $\sum_k c_k=2$, such that $c_0=c_2=\frac{1}{2}$, $c_1=1$, and all other c_k's are zero;

and $\phi_0(x)=\begin{cases}x, & 0\le x<1\\ 2-x, & 1\le x\le 2\end{cases}$

Solution: Since $\phi_0(x) = \begin{cases} x, & 0 \le x < 1 \\ 2-x, & 1 \le x \le 2 \end{cases}$

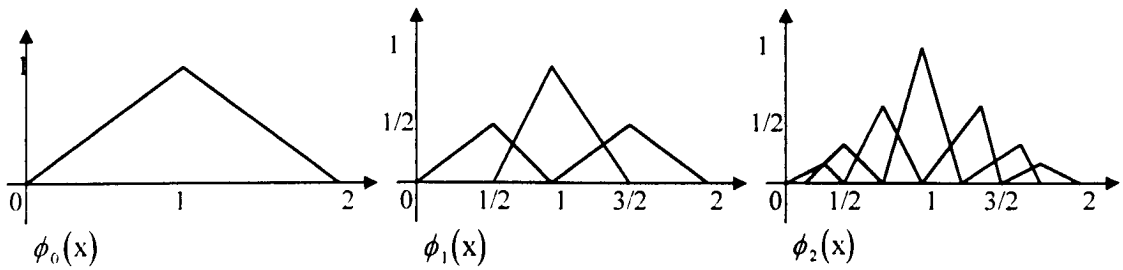
then $\phi_1(x) = \frac{1}{2}\phi_0(2x) + \phi_0(2x-1) + \frac{1}{2}\phi_0(2x-2)$

and $\phi_2(x) = \frac{1}{2}\phi_1(2x) + \phi_1(2x-1) + \frac{1}{2}\phi_1(2x-2).$

$$= \frac{1}{4}\phi_0(4x) + \frac{1}{2}\phi_0(4x-1) + \frac{3}{4}\phi_0(4x-2) + \phi_0(4x-3) + \frac{3}{4}\phi_0(4x-4)$$

$$+ \frac{1}{2}\phi_0(4x-5) + \frac{1}{4}\phi_0(4x-6).$$

Graphs of the functions $\phi_0(x)$, $\phi_1(x)$, and $\phi_2(x)$ are shown below:



It is to be noted that $\phi_2(x)$ is symmetric about $x = 1$ and its integral equals 1. This can be proved by taking the integral of $\phi_2(x)$ given in the above relation that

$$\frac{1}{2}\int_0^2 \phi_2(x)dx = \frac{1}{2}\left(\frac{1}{8} + \frac{2}{8} + \frac{3}{8} + \frac{4}{8} + \frac{3}{8} + \frac{2}{8} + \frac{1}{8}\right) = 1$$

The following sequence is important and can be used to form wavelet.

$$\frac{1}{2}\left\{\frac{1}{8}, \frac{2}{8}, \frac{3}{8}, \frac{4}{8}, \frac{3}{8}, \frac{2}{8}, \frac{1}{8}\right\}$$

Similarly, $\frac{1}{4}\left\{\frac{1}{16}, \frac{2}{16}, \frac{3}{16}, \frac{4}{16}, \frac{5}{16}, \frac{6}{16}, \frac{7}{16}, \frac{8}{16}, \frac{7}{16}, \frac{6}{16}, \frac{5}{16}, \frac{4}{16}, \frac{3}{16}, \frac{2}{16}, \frac{1}{16}\right\}$.

It can be noted that 2^{n-1} gives the energy of the function $\phi_n(x)$.

General Iterative Formula to Compute Wavelet Coefficients.

$$\phi^1(x) = c_{10}\phi^0{}_{10}(2x) + c_{11}\phi^0{}_{11}(2x-1) + \ldots + c_{1i}\phi^0{}_{1i}(2x-i)$$

$$= \sum_{i=0}^{n-1} c_{1i}\phi^0(2x-i), \text{ where n is the maximum index for non-zero coefficients } c_{1i}\text{'s.}$$

$$\phi^2(x) = c_{10}\phi^1{}_{10}(2x) + c_{11}\phi^1{}_{11}(2x-1) + \ldots + c_{1i}\phi^1{}_{1i}(2x-i)$$

$$= c_{10}\left(\sum_{i=0}^{n-1} c_{1i}\phi^0(2^2x-i)\right) + c_{11}\left(\sum_{i=0}^{n-1} c_{1i}\phi^0(2^2x-i-1)\right)+\ldots+c_{1k}\left(\sum_{i=0}^{n-1} c_{1i}\phi^0(2^2x-i-k)\right)$$

$$= \sum_{k=0}^{n-1} c_{1k}\left(\sum_{i=0}^{n-1} c_{1i}\phi^0(2^2x-i-k)\right)$$

$$\phi^3(x) = \sum_{k=0}^{n-1} c_{1k}\left(\sum_{i=0}^{n-1} c_{1i}\phi^0(2^3x-i-k)\right)$$

.

.

.

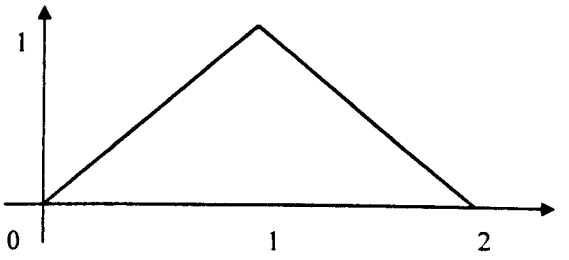
$$\phi^j(x) = \sum_{k=0}^{n-1} c_{1k}\left(\sum_{i=0}^{n-1} c_{1i}\phi^0(2^jx-i-k)\right)$$

6.17 Developing and Elaborating some Wavelets Mathematically

(a) Defining a new $\psi(x)$ function from the hat function $\phi(x)$

Consider the hat function

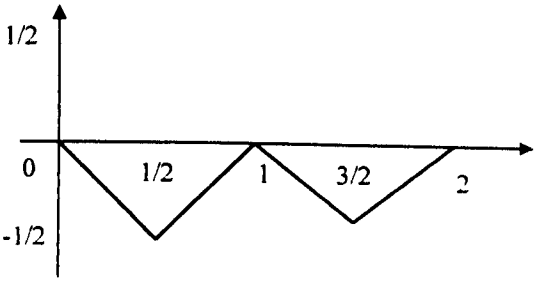
$$\phi(x) = \begin{cases} x, & 0 \le x < 1 \\ 2-x, & 1 \le x < 2 \end{cases}$$



We know that for $c_1=1$, $c_0=c_2=1/2$, the solution of dilation equation $\phi(x) = \sum_k c_k \phi(2x-k)$ is the hat function. We can write $\phi(x) = \frac{1}{2}\phi(2x) + \phi(2x-1) + \frac{1}{2}\phi(2x-2)$, where the mirror images of $\frac{1}{2}\phi(2x)$ and $\frac{1}{2}\phi(2x-2)$ are given by

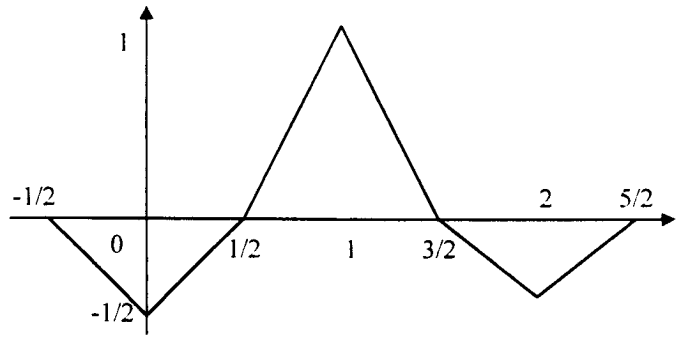
$$-\frac{1}{2}\phi(2x) = \begin{cases} -x, & 0 \le x < 1/2 \\ x-1, & 1/2 \le x < 1 \end{cases}$$

$$-\frac{1}{2}\phi(2x-2) = \begin{cases} 1-x, & 1 \le x < 3/2 \\ x-2, & 3/2 \le x \le 2 \end{cases}$$



By shifting $-\frac{1}{2}\phi(2x)$ to left by ½ and $-\frac{1}{2}\phi(2x-2)$ to right by ½, we can define

$$\psi(x) = \begin{cases} -x-1/2, & -1/2 \le x < 0 \\ x-1/2, & 0 \le x < 1/2 \\ 2x-1, & 1/2 \le x < 1 \\ 3-2x, & 1 \le x < 3/2 \\ 3/2-x, & 3/2 \le x < 2 \\ x-5/2, & 2 \le x < 5/2 \end{cases}$$



(b) Defining $\psi(x)$ from the hat function $\phi(x)$

The wavelet function ψ, related to a scaling function ϕ, can be derived, as follows:-

The hat function is given by

$$\phi(x) = \begin{cases} x, & 0 \le x < 1 \\ 2-x, & 1 \le x < 2 \end{cases}$$

We define $\psi(x)$ by the formula

$$\psi(x) = \sum_k (-1)^k \bar{c}_{1-k} \phi(2x-k)$$

Putting k=1,0, and -1, we have

$$\psi(x) = -1\bar{c}_0\phi(2x-1) + \bar{c}_1\phi(2x) - 1\bar{c}\phi(2x+1)$$

$$= -c_0\phi(2x-1) + c_1\phi(2x) - c_2\phi(2x+1)$$

where

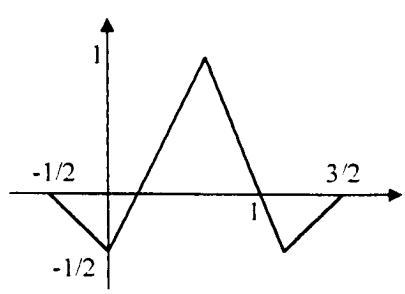
$$-c_0\phi(2x-1) = -\frac{1}{2}\begin{cases} 2x-1, & 0 \le 2x-1 < 1 \\ 2-(2x-1), & 1 \le 2x-1 < 2 \end{cases} = \begin{cases} -x+1/2, & 1/2 \le x < 1 \\ -3/2+x, & 1 \le x < 3/2 \end{cases}$$

$$c_1\phi(2x) = \begin{cases} 2x, & 0 \le 2x < 1 \\ 2-2x, & 1 \le 2x < 2 \end{cases} = \begin{cases} 2x, & 0 \le x < 1/2 \\ 2-2x, & 1/2 \le 2x < 1 \end{cases}$$

$$-c_2\phi(2x+1) = -\frac{1}{2}\begin{cases} 2x+1, & 0 \le 2x+1 < 1 \\ 2-(2x+1), & 1 \le 2x+1 < 2 \end{cases} = \begin{cases} -x-1/2, & -1/2 \le x < 0 \\ -1/2+x, & 0 \le x < 1/2 \end{cases}$$

By adding, we have

$$\psi(x) = \begin{cases} -x-1/2, & -1/2 \le x < 0 \\ -1/2+3x, & 0 \le x < 1/2 \\ -3x+5/2, & 1/2 \le x < 1 \\ x-3/2, & 1 \le x < 3/2 \end{cases}$$



© Defining D_4 Scaling Function ϕ from the Box Function

The box function is given by

$$\phi(x) = \begin{cases} 1, & 0 \le x < 1 \\ 0, & \text{otherwise} \end{cases}$$

To solve the refinement equation $\phi(x) = \sum_k c_k \phi(2x - k)$, let $c_0=\sqrt{2}(1+\sqrt{3})/4$, $c_1=\sqrt{2}(3+\sqrt{3})/4$, $c_2=\sqrt{2}(3-\sqrt{3})/4$, and $c_3=\sqrt{2}(1-\sqrt{3})/4$. So that

$$c_0\phi(2x) = \frac{\sqrt{2}}{4}(1+\sqrt{3})\begin{cases} 1, & 0 \le 2x < 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} \sqrt{2}(1+\sqrt{3})/4, & 0 \le x < 1/2 \\ 0, & \text{otherwise} \end{cases}$$

$$c_1\phi(2x-1) = \begin{cases} \sqrt{2}(3+\sqrt{3})/4, & 0 \le 2x-1 < 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} \sqrt{2}(3+\sqrt{3})/4, & 1/2 \le x < 1 \\ 0, & \text{otherwise} \end{cases}$$

$$c_2\phi(2x-2) = \begin{cases} \sqrt{2}(3-\sqrt{3})/4, & 0 \le 2x-2 < 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} \sqrt{2}(3-\sqrt{3})/4, & 1 \le x < 3/2 \\ 0, & \text{otherwise} \end{cases}$$

$$c_3\phi(2x-3) = \begin{cases} \sqrt{2}(1-\sqrt{3})/4, & 0 \le 2x-3 < 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} \sqrt{2}(1-\sqrt{3})/4, & 3/2 \le x < 2 \\ 0, & \text{otherwise} \end{cases}$$

Adding, we have

$$\phi(x) = \begin{cases} \sqrt{2}(1+\sqrt{3})/4, & 0 \le x < 1/2 \\ \sqrt{2}(3+\sqrt{3})/4, & 1/2 \le 2 < 1 \\ \sqrt{2}(3-\sqrt{3})/4, & 1 \le x < 3/2 \\ \sqrt{2}(1-\sqrt{3})/4, & 3/2 \le x < 2 \end{cases}$$

If we digitise this function ϕ such that

$$\{\phi(x)\} = \left\{\sqrt{2}(1+\sqrt{3})/4,\ \sqrt{2}(3+\sqrt{3})/4,\ \sqrt{2}(3-\sqrt{3})/4,\ \sqrt{2}(1-\sqrt{3})/4\right\}.$$

then $\sum|\phi(x)|^2 = 1$. Hence $\{\phi(x)\}$ is a scaling function.

Differently, if we define $\{\Phi(x)\}$ as

$$\{\Phi(x)\} = \left\{\sqrt{2}(-1+\sqrt{3})/4,\ \sqrt{2}(-3+\sqrt{3})/4,\ \sqrt{2}(3-\sqrt{3})/4,\ \sqrt{2}(1-\sqrt{3})/4\right\}$$

then $\sum|\Phi(x)|^2 = 1$ and $\sum\Phi(x) = 0$. Therefore $\{\Phi(x)\}$ qualifies as a wavelet.

(d) Defining D_4 Wavelet $\psi(x)$ from the Box Function

To solve the refinement equation $\psi(x) = \sum_k (-1)^k \bar{c}_{1-k} \phi(2x-k)$ for $c_0 = \frac{1}{4\sqrt{2}}(1+\sqrt{3})$, $c_1 = \frac{1}{4\sqrt{2}}(3+\sqrt{3})$, $c_2 = \frac{1}{4\sqrt{2}}(3-\sqrt{3})$, and $c_3 = \frac{1}{4\sqrt{2}}(1-\sqrt{3})$; let us put k=1,0,-1,-2.

So that

$$\psi(x) = -\bar{c}_0\phi(2x-1) + \bar{c}_1\phi(2x) - \bar{c}_2\phi(2x+1) + \bar{c}_3\phi(2x+2)$$

$$= -c_0\phi(2x-1) + c_1\phi(2x) - c_2\phi(2x+1) + c_2\phi(2x+2)$$

where

$$-c_0\phi(2x-1) = -\frac{1}{4\sqrt{2}}(1+\sqrt{3})\begin{cases} 1, & 0 \le 2x-1 < 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} -(1+\sqrt{3})/4\sqrt{2}, & 1/2 \le x < 1 \\ 0, & \text{otherwise} \end{cases}$$

$$c_1\phi(2x) = \frac{1}{4\sqrt{2}}(3+\sqrt{3})\begin{cases} 1, & 0 \le 2x < 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} (3+\sqrt{3})/4\sqrt{2}, & 0 \le x < 1/2 \\ 0, & \text{otherwise} \end{cases}$$

$$-c_2\phi(2x+1) = -\frac{1}{4\sqrt{2}}(3-\sqrt{3})\begin{cases} 1, & 0 \le 2x+1 < 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} -(3-\sqrt{3})/4\sqrt{2}, & -1/2 \le x < 0 \\ 0, & \text{otherwise} \end{cases}$$

and

$$c_3\phi(2x+2) = \frac{1}{4\sqrt{2}}(1-\sqrt{3})\begin{cases} 1, & 0 \le 2x+2 < 1 \\ 0, & \text{otherwise} \end{cases} = \begin{cases} (1-\sqrt{3})/4\sqrt{2}, & -1 \le x < -1/2 \\ 0, & \text{otherwise} \end{cases}$$

Adding, after re-arranging, we have

$$\psi(x) = \begin{cases} (1-\sqrt{3})/4\sqrt{2}, & -1 \le x < -1/2 \\ -(3-\sqrt{3})/4\sqrt{2}, & -1/2 \le x < 0 \\ (3+\sqrt{3})/4\sqrt{2}, & 0 \le x < 1/2 \\ -(1+\sqrt{3})/4\sqrt{2}, & 1/2 \le x < 1 \end{cases}$$

If we digitise this ψ such that

$$\{\psi(x)\} = \left\{(1-\sqrt{3})/4\sqrt{2},\ -(3-\sqrt{3})/4\sqrt{2},\ (3+\sqrt{3})/4\sqrt{2},\ -(1+\sqrt{3})/4\sqrt{2}\right\},$$

then $\sum \psi(x) = 0$ and $\sum |\psi(x)|^2 = 1$. Hence ψ is a wavelet. This wavelet can be applied as a window to compute wavelet transform and its inverse.

(e) Associating Multiresolution with D4 Scaling Function

The box function is given by

$$\phi(x) = \begin{cases} 1, & 0 \le x < 1 \\ 0, & \text{otherwise} \end{cases}$$

Consider the dilation equation $\phi(x) = 2^{j/2} \sum_{k=0}^{3} c_k \phi(2^j x - k)$, with support $I=[2^{-j}k, 2^{-j}(k+1)]$;

where $c_0=\sqrt{2}(1+\sqrt{3})/4$, $c_1=\sqrt{2}(3+\sqrt{3})/4$, $c_2=\sqrt{2}(3-\sqrt{3})/4$, and $c_3=\sqrt{2}(1-\sqrt{3})/4$.

So that $\phi = 2^{j/2}\{c_0\phi(2^j x) + c_1\phi(2^j x - 1) + c_2\phi(2^j x - 2) + c_2\phi(2^j x - 3)\}$

or

$$\phi(x) = 2^{j/2} \begin{cases} c_0\phi_{j0}(x) : 0 \le x < 1/2^j \\ c_1\phi_{j1}(x) : 1/2^j \le x < 2/2^j \\ c_2\phi_{j2}(x) : 2/2^j \le x < 3/2^j \\ c_3\phi_{j3}(x) : 3/2^j \le x < 4/2^j \end{cases} = 2^{j/2} \begin{cases} \sqrt{2}(1+\sqrt{3})/4, & 0 \le x < 1/2^j \\ \sqrt{2}(3+\sqrt{3})/4, & 1/2^j \le 2 < 2/2^j \\ \sqrt{2}(3-\sqrt{3})/4, & 2/2^j \le x < 3/2^j \\ \sqrt{2}(1-\sqrt{3})/4, & 3/2^j \le x < 4/2^j \end{cases}$$

For j=1, 2,3; we have

$$\phi(x) = 2^{1/2} \begin{cases} c_0\phi_{10}(x) : 0 \le x < 1/2^1 \\ c_1\phi_{11}(x) : 1/2^1 \le x < 2/2^1 \\ c_2\phi_{12}(x) : 2/2^1 \le x < 3/2^1 \\ c_3\phi_{13}(x) : 3/2^1 \le x < 4/2^1 \end{cases} = 2^{1/2} \begin{cases} \sqrt{2}(1+\sqrt{3})/4, & 0 \le x < 1/2 \\ \sqrt{2}(3+\sqrt{3})/4, & 1/2 \le 2 < 1 \\ \sqrt{2}(3-\sqrt{3})/4, & 1 \le x < 3/2 \\ \sqrt{2}(1-\sqrt{3})/4, & 3/2 \le x < 2 \end{cases}$$

$$\phi(x) = 2^{2/2} \begin{cases} c_0\phi_{20}(x) : 0 \le x < 1/2^2 \\ c_1\phi_{21}(x) : 1/2^2 \le x < 2/2^2 \\ c_2\phi_{22}(x) : 2/2^2 \le x < 3/2^2 \\ c_3\phi_{23}(x) : 3/2^2 \le x < 4/2^2 \end{cases} = 2^{2/2} \begin{cases} \sqrt{2}(1+\sqrt{3})/4, & 0 \le x < 1/4 \\ \sqrt{2}(3+\sqrt{3})/4, & 1/4 \le 2 < 1/2 \\ \sqrt{2}(3-\sqrt{3})/4, & 1/2 \le x < 3/4 \\ \sqrt{2}(1-\sqrt{3})/4, & 3/4 \le x < 1 \end{cases}$$

$$\phi_{3k}(x) = 2^{3/2} \begin{cases} c_0\phi_{30}(x) : 0 \le x < 1/2^3 \\ c_1\phi_{31}(x) : 1/2^3 \le x < 2/2^3 \\ c_2\phi_{32}(x) : 2/2^3 \le x < 3/2^3 \\ c_3\phi_{33}(x) : 3/2^3 \le x < 4/2^3 \end{cases} = 2^{3/2} \begin{cases} \sqrt{2}(1+\sqrt{3})/4, & 0 \le x < 1/8 \\ \sqrt{2}(3+\sqrt{3})/4, & 1/8 \le 2 < 1/4 \\ \sqrt{2}(3-\sqrt{3})/4, & 1/4 \le x < 3/8 \\ \sqrt{2}(1-\sqrt{3})/4, & 3/8 \le x < 1/2 \end{cases}$$

In the similar way we can have higher resolutions.

Continuous Wavelet Transform and its Application

The main objective of this chapter is to investigate the Continuous Wavelet Transform (CWT) and its application for noise reduction and data compression. A magical behaviour of zoom-out plus noise reducing wavelet transform that the window-width automatically decreases on the increase of data expansion and the behaviour of zoom-in plus noise reducing wavelet transform that the window-width automatically increases on the increase of data compression, will be discussed in detail and performance of the transforms will be checked against the self developed software. Unlike the case of continuous wavelets, the concept of expanding a digital wavelet is complicated to visualise. It will therefore be focused in detail in Section(7.2) along with introducing a new wavelet.

7.1 Some Basics of Continuous Wavelet Transform

It is expressed in [8] that the continuous wavelet transform can be written as

$$CWT_f(b.a) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(x)\psi(\frac{x-b}{a})dx . \qquad (7.1.1)$$

where $a > 0$ and b are called the scaling and the shift parameters respectively. It is advantageous to choose $a = 2^{-j}$ and $b = ak$, $k \in Z$ [7]. In this way "the smaller the value of a the larger the magnification is, the more detail we want to catch" and "small $a \Rightarrow$ large magnification $\Rightarrow$ small step $\Rightarrow$ small b. Thus choosing $b = ak$ is natural".

Let $[0,l[$ denote the domain of both the input function f and the wavelet function ψ and let m, where $m < l$, be the number of non-zero elements of ψ that constitute a window, then by using non-negative indices Equation(7.1.1) can be written as

$$CWT_f(k,j) = \frac{1}{\lambda} \int_{2^{-j}k}^{2^{-j}k+m-1} f(x)\psi(2^j x - k)dx \text{ , where } k = 0,1,....,2^j(l-m) , \qquad (7.1.2)$$

and its digitised version is

$$f_j(k) = \frac{2^{j/2}}{\lambda} \sum_{i=1}^{l} f(i-1)\psi(2^j i - k) \text{ , where } \quad k = \frac{m}{2}+1, \frac{m}{2}, ..., 0,1,....,2^j(l-\frac{m}{2}) \qquad (7.1.3)$$

It should be noted that in Equations(7.1.3) the values of k other than $k = 0,1,.....2^j(l-m)$ are to apply the zero-padding scheme. By ignoring the zero-padding, we have

$$f_j(k) = \frac{2^{j/2}}{\lambda}\sum_{i=1}^{l} f(i-1)\psi(2^j i - k), \text{ where } k = 0,1,.....,2^j(l-m) \qquad (7.1.4)$$

This system of equations expresses that the window of ψ (having basic width equals m) is automatically

(i) compressed by a factor of $1/2^j$, $j > 0$, moves with step-size equals 1 and produces 2^j elements in each step and the transform is called zoom-out wavelet transform.

(ii) expanded by a factor 2^j, $j > 0$, moves with step-size equals 2^j and produces 1 element is each step and the transform is called zoom-in wavelet transform.

Orthogonality of Wavelets:

A function ψ is called an orthonormal wavelet if its family $\left\{\psi_{j,k} = 2^{j/2}\psi(2^j x - k),\quad j,k \in Z\right\}$ form a system of orthonormal wavelets; that is, if

$$\left\langle \psi_{j,k}, \psi_{l,m} \right\rangle = \delta_{j,l}.\delta_{k,m},\quad j,k,l,m \in Z.$$

It has practically been observed that the mutual scalar product of different family members is not required in the application of CWT for noise reduction and data compression. However, the family of wavelets should be orthogonal. That is, the family members should satisfy

$$\left\langle \psi_{j,k}, \psi_{l,m} \right\rangle = 1, \text{ when } j = l \text{ and } k = m.$$

Let us consider, for example, a basic wavelet ψ defined as

$$\{\psi(i)\}_{i\in Z} = \frac{1}{\sqrt{60}}[...,0,1,2,3,4,4,3,2,1,0,...]. \qquad (7.1.5)$$

For j=0, we have two adjacent family members

$$\psi_{0,k} = \frac{1}{\sqrt{60}}[...,0,1,2,3,4,4,3,2,1,0,...]$$

$$\psi_{0,k+1} = \frac{1}{\sqrt{60}}[...,0,0,1,2,3,4,4,3,2,1,0,...]$$

which are orthogonal since $\left\langle \psi_{0,k}, \psi_{0,k} \right\rangle = \left\langle \psi_{0,k+1}, \psi_{0,k+1} \right\rangle = 1$ but are not arthonormal since

$$\langle \psi_{0,k}, \psi_{0,k+1} \rangle \neq 0 .$$

The family given in Equation(7.1.5) can be used as it were orthonormal by avoiding to compute their mutual scalar product throughout the application.

7.2 A New Wavelet with a New Defining Style and the Automatic Nature of CWT

After success replacing the task of average smoothing filter from sequential to parallel processing by developing an ANN shown in Figure(1.2.1), a new wavelets was developed is section(9.2) with the purpose of avoiding the repeated application of the ANN. This idea lead us to develop a new wavelet given in Equation(7.2.1). Moreover, the mathematical expression $\psi(2^j i - k)$ shows evidently that the new wavelet is the identity wavelet and it is, therefore, the simplest possible one.

If $t \in \Re$ and $m \in Z^+$, then we can define a basic wavelet window function ψ as given below:

$$\psi(t) = \frac{1}{\lambda} \begin{cases} 0, & t \leq 0 \\ t, & 0 < t \leq m/2 \\ m - t, & m/2 < t < m \\ 0, & t \geq m \end{cases} \qquad (7.2.1)$$

where λ denotes the energy of ψ. The purposes of presenting this wavelet will become clear in the subsequent sections and chapters. The concept of automatic wavelet compression/expansion during the wavelet transform is concern with its family members.

Concept of Compression and/or Expansion of Continuous and Digital Wavelets:

There is a confusing difference between the compression/expansion of continuous and digital functions (wavelets). The wavelet defined in Equation(7.2.1) does not reflect any of the two concepts. It is therefore necessary to define its family members in two ways; one for continuous and the other for digital wavelets.

Case-1: A continuous family of the basic wavelet is defined as

$$\psi(2^j t) = \frac{1}{\lambda} \begin{cases} 0, & t \leq 0 \\ 2^j t, & 0 < t \leq m/2 \\ 2^j (m - t) & m/2 < t < m \\ 0, & t \geq m \end{cases} \qquad (7.2.2)$$

Observe that the compression of the wavelet corresponds to positive values of j and is done by simply down sampling the wavelet. This property of wavelets is required advantageously for zoom-out plus noise reduction wavelet transform. See Section(7.4) for details. But on the other hand, the expansion of the wavelet corresponds to negative values of j and is done by recruiting additional non available real numbers between the existing ones. Although the resolution of the wavelet is increased but the width of (domain of definition, that is, interval $]0,m[$) the wavelet remains the same. This drawback should be visualised for the digitised version of the wavelet where we will have to compute the additional numbers from their neighbouring ones. See for example Section(8.1) where the additional numbers of similar nature are computed for the signals. Further, as a good mathematical example with graphical support see Section(6.16). Thus the interpretation of $\frac{1}{\lambda}\{\psi(2^j x)\}$ given as Equation(7.2.2) is miss-leading towards the concept of automatic wavelet expansion during zoom-in plus noise reduction wavelet transform. For example,

for m=8 the function $\frac{1}{\lambda}\{\psi(2^j i)\}_{i\in Z}$ defined in Equation(7.2.2) is given below:

(i) when j=-1, then

$$\frac{1}{\lambda}\left\{\psi(\frac{i}{2})\right\}_{i\in Z} = \frac{1}{\sqrt{60}}\{...,0,1/2,1,3/2,2,5/2,3,7/2,4,4,7/2,3,5/2,2,3/2,1,1/2,0,...\}$$

(ii) when j=-2, then

$$\frac{1}{\lambda}\left\{\psi(\frac{i}{2^2})\right\}_{i\in Z} = \frac{1}{\sqrt{60}}\{..,0,1/4,1/2,3/4,1,5/4,3/2,7/4,2,9/4,5/2,11/4,3,13/4,7/2,15/4,4,$$

$$4,15/4,7/2,13/4,3,11/4,5/2,9/4,2,7/4,3/2,5/4,1,3/4,1/2,1/4,0,...\}$$

This shows that the automatic expansion of the wavelet requires unavailable additional values to be inserted between each pair of elements of the digitised wavelet. This drawback of the definition is miss leading and produces a conceptual confusion.

But if we divide and multiply by 2 on both sides in the case (i) and (ii) above then we have for j=-1, -2, the wavelets

$$\frac{1}{2}\frac{1}{\sqrt{60}}\{....0,1,2,3,4,5,6,7,8,8,7,6,5,4,3,2,1,0,...\}\text{, and}$$

$$\frac{1}{4}\frac{1}{\sqrt{60}}\{..,1,2,3,4,5,6,7,8,9,10,11,12,13,14,16,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0,...\}$$

respectively. This idea leads us to a new style of defining the wavelets as given below in case-2.

Case-2: A digital family of a basic wavelet for zoom-in wavelet transform should be defined as

$$\psi(2^j t)=\frac{1}{\lambda}\begin{cases}0, & 2^{-j}t\le 0\\ t, & 0<2^{-j}t\le m/2\\ m-t & m/2<2^{-j}t<m\\ 0, & 2^{-j}t\ge m\end{cases}$$

Or equivalently,

$$\psi(2^j t)=\frac{1}{\lambda}\begin{cases}0, & t\le 0\\ t, & 0<t\le m/2^{j+1}\\ m-t & m/2^{j+1}<t<m/2^{j}\\ 0, & t\ge m/2^{j}\end{cases}\qquad(7.2.3)$$

Observe that the compression of the wavelet corresponds to positive values of j and is done by reducing the width of the domain of definition which is the open interval $]0,m[$. This drawback is miss-leading for zoom-out plus noise reduction wavelet transform. Either a researcher may be struck off at this stage or the transform will produced the expansion by repeating 2^j times the weighted sum in each window placing. Consequently, the output will require data smoothing subsequently. See for example section Software(8.1). On the other hand, the expansion of the wavelet corresponds to negative values of j and is done by increasing the width of the interval from $]0,m[$ to $]0,m/2^{-j}[$. This interpretation of $\frac{1}{\lambda}\{\psi(2^j x)\}$ given in Equation(7.2.3) is blessing for noise reduction and data compression wavelet transform. For a good mathematical example with graphical support see Section(6.15). There, the family of rectangles provides us the concept of relative bandwidth while here the family of Gaussian-like wavelets provides the concept of constant relative bandwidth as the height of the basic

wavelet and all of its family members is the same. Let us now consider the dependency of the width on the resolution parameter to observe that for m=32 the function $\frac{1}{\lambda}\{\psi(2^j i)\}_{i\in Z}$ is given below:

(i) when j=0, then $\frac{1}{\lambda}\{\psi(2^0 i)\}_{i\in Z} = \frac{1}{\sqrt{2736}}\{ \ldots,0,\ 1,\ 2,\ 3,\ldots..,15,\ 16,\ 15,\ldots..,3,\ 2,\ 1,\ 0,\ldots \}$

(ii) when j=1, then $\frac{1}{\lambda}\{\psi(2^1 i)\}_{i\in Z} = \frac{1}{\sqrt{1376}}\{ \ldots,0,\ 2,\ 4,\ 6,\ldots..,14,\ 16,\ 14,\ldots..,6,\ 4,\ 2,\ 0,\ldots \}$

$$= \frac{2}{\sqrt{1376}}\{ \ldots,0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 7,\ 6,\ 5,\ 4,\ 3,\ 2,\ 1,\ 0,\ldots \}$$

(iii) when j=2, then

$$\frac{1}{\lambda}\{\psi(2^2 i)\}_{i\in Z} = \frac{1}{\sqrt{704}}\{ \ldots,0,\ 4,\ 8,\ 12,\ 16,\ 12,\ 8,\ 4,\ 0,\ldots \} = \frac{4}{\sqrt{704}}\{ \ldots,0,\ 1,\ 2,\ 3,\ 4,\ 3,\ 2,\ 1,\ 0,\ldots \}$$

In general, we have

$$\frac{1}{\lambda}\{\psi(2^j i)\}_{i\in Z} = \frac{2^{j/2}}{\lambda}\left\{ \ldots,0,1,2,3,\ldots..,\frac{m/2}{2^j}-1,\ \frac{m/2}{2^j},\ \frac{m/2}{2^j}-1,\ldots..,3,2,1,0,\ldots \right\}$$

for m=8 the function $\frac{1}{\lambda}\{\psi(2^j i)\}_{i\in Z}$ is given below:

(i) when j=0, then $\frac{1}{\lambda}\{\psi(2^0 i)\}_{i\in Z} = \frac{1}{\sqrt{44}}\{ \ldots0,\ 1,\ 2,\ 3,\ 4,\ 3,\ 2,\ 1,0,\ldots \}$

(ii) when j=-1, then $\frac{1}{\lambda}\left\{\psi(\frac{i}{2})\right\}_{i\in Z} = \frac{1}{2}\frac{1}{\sqrt{344}}\{\ldots,0,1,2,3,4,5,6,7,8,7,6,5,4,3,2,1,0,\ldots\}$

(iii) when j=2, then

$$\frac{1}{\lambda}\left\{\psi(\frac{i}{2^2})\right\}_{i\in Z} = \frac{1}{4}\frac{1}{\sqrt{2736}}\{..,1,2,3,4,5,6,7,8,9,10,11,12,13,14,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0,\ldots$$

In general, we have

$$\frac{1}{\lambda}\{\psi(2^j i)\}_{i\in Z} = \frac{1}{2^j}\frac{1}{\lambda}\left\{ \ldots,0,1,2,\ldots..,\frac{m/2}{2^j}-1,\ \frac{m/2}{2^j},\ \frac{m/2}{2^j}-1,\ldots..,2,1,0,\ldots \right\}$$

In order to have width equals integral power of 2, we define the family of ψ slightly different as given below:

$$\psi(2^j t) = \frac{1}{\lambda}\begin{cases} 0, & t \le 0 \\ t, & 0 < t \le m/2^{j+1} \\ m-(t-1) & m/2^{j+1} < t \le m/2^j \\ 0, & t \ge m/2^j \end{cases} \qquad (7.2.4)$$

In this case for m=16 and $j \ge 0$, we have

$$\frac{1}{\lambda}\{\psi(2^0 i)\}_{i\in Z} = \frac{1}{\sqrt{408}}\{\ ...,0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 8,\ 7,\ 6,\ 5,\ 4,\ 3,\ 2,\ 1,\ 0,...\ \},$$

$$\frac{1}{\lambda}\{\psi(2^1 i)\}_{i\in Z} = \frac{2}{\sqrt{60}}\{\ ...,0,\ 1,\ 2,\ 3,\ 4,\ 4,\ 3,\ 2,\ 1,\ 0,...\ \},$$

$$\frac{1}{\lambda}\{\psi(2^2 i)\}_{i\in Z} = \frac{4}{\sqrt{10}}\{\ ...,0,\ 1,\ 2,\ 2,\ 1,\ 0,...\ \}.$$

While for m=8 and $j \le 0$, we have

$$\frac{1}{\lambda}\{\psi(2^0 i)\}_{i\in Z} = \frac{1}{\sqrt{60}}\{\ ...0,1,\ 2,\ 3,\ 4,\ 4,\ 3,\ 2,\ 1,0...\ \}$$

$$\frac{1}{\lambda}\{\psi(2^{-1} i)\}_{i\in Z} = \frac{1}{2}\frac{1}{\sqrt{60}}\{...,0.1.2,3,4,5,6,7.8,8,7,6,5,4,3,2,1,0,...\}$$

$$\frac{1}{\lambda}\{\psi(2^{-2} i)\}_{i\in Z} = \frac{1}{4}\frac{1}{\sqrt{60}}\{...1.2.3.4.5.6.7.8.9.10.11.12.13.14.16.16.15.14.13.12.11.10.9.8.7.6.5.4.3.2.1.0,...\}$$

7.3 A Wavelet Transform for Noise Reduction

By setting m=8 and j=0 in Equation(7.2.4), we have the wavelet

$$\frac{1}{\lambda}\{\psi(2^0 i)\}_{i\in Z} = \frac{1}{\sqrt{60}}\{..., 0, 1, 2, 3, 4, 4, 3, 2, 1, 0, ...\}$$

Using this wavelet in Equations(7.1.4) which is given as

$$f_j(k) = \frac{2^{j/2}}{\lambda}\sum_{i=1}^{l} f(i-1)\psi(2^j i - k), \text{ where } k = 0,1,....,2^j(l-m).$$

We can compute the Noise Reducing Wavelet Transform as given below:

$$f_0(0) = \frac{1}{\sqrt{60}}\{f(0)\psi(1) + f(1)\psi(2) + ... + f(7)\psi(8) + ... + f(31)\psi(32)\}$$

$$= \frac{1}{\sqrt{60}}\{1f(0) + ... + 4f(3) + 4f(4) + ... + 1f(7) + 0f(8) + ... + 0f(31)\}$$

$$= \frac{1}{\sqrt{60}}\{1f(0)+...+4f(3)+4f(4)+...+1f(7)\}$$

$$f_0(1) = \frac{1}{\sqrt{60}}\{f(0)\psi(0)+f(1)\psi(1)+...+f(8)\psi(8)+...+f(31)\psi(31)\}$$

$$= \frac{1}{\sqrt{60}}\{0f(0)+1f(1)+...+4f(3)+4f(4)+...+1f(8)+0f(9)+....+0f(31)\}$$

$$= \frac{1}{\sqrt{60}}\{1f(1)+...+4f(4)+4f(5)+...+1f(8)\}$$

In the same way one can compute $f_0(k)$, $k = 2,3,...,l-m$.

It is clear that the window moves along the signal with step-size equals 1 and one element is produced in each window placing. This case is in accordance with the modified definition of Wavelet Transform presented in Section(8.2).

7.4 Automatic Wavelet Compression and Data Expansion (multiresolution zoom-out DWT)

The drawback free style of defining wavelets for zoom-out plus noise reduction wavelet transform is expressed in Section(7.2) and is given in Relation(7.2.2). Using this style with m=8 and j=0 in Relation(7.2.4), we have the wavelet

$$\frac{1}{\lambda}\{\psi(2^0 i)\}_{i\in Z} = \frac{1}{\sqrt{60}}\{..., 0, 1, 2, 3, 4, 4, 3, 2, 1, 0, ...\}$$

which will be used as the basic wavelet in showing automatic wavelet compression and data expansion plus noise reduction.

Using this wavelet basically and setting j=1 in transform Equations(7.1.4), we have

$$f_1(0) = \frac{\sqrt{2}}{\sqrt{60}}\{f(0)\psi(2)+f(1)\psi(4)+f(2)\psi(6)+f(3)\psi(8)+...+f(31)\psi(64)\}$$

$$= \frac{1}{\sqrt{30}}\{2f(0)+4f(1)+3f(2)+1f(3)+0f(8)+....+0f(31)\}$$

$$= \frac{1}{\sqrt{30}}\{2f(0)+4f(1)+3f(2)+1f(3)\}$$

$$f_1(1) = \frac{1}{\sqrt{30}}\{f(0)\psi(1)+f(1)\psi(3)+f(2)\psi(5)+f(3)\psi(7)+...+f(31)\psi(63)\}$$

$$= \frac{1}{\sqrt{30}}\{1f(0)+3f(1)+4f(2)+2f(3)+0f(4)+....+0f(31)\}$$

$$= \frac{1}{\sqrt{30}}\{1f(0)+3f(1)+4f(2)+2f(3)\}$$

It is clear that when the value of j is increased then the size of the window is decreased and its energy is maintained.

$$f_1(2) = \frac{\sqrt{2}}{\sqrt{60}}\{f(0)\psi(0)+f(1)\psi(2)+...+f(4)\psi(8)+...+f(31)\psi(62)\}$$

$$= \frac{1}{\sqrt{30}}\{0f(0)+2f(1)+4f(2)+3f(3)+1f(4)+0f(5)+...+0f(31)\}$$

$$= \frac{1}{\sqrt{30}}\{2f(1)+4f(2)+3f(3)+1f(4)\}$$

$$f_1(3) = \frac{\sqrt{2}}{\sqrt{60}}\{f(0)\psi(-1)+f(1)\psi(1)+f(2)\psi(3)+f(3)\psi(5)+f(4)\psi(7)+...+f(31)\psi(61)\}$$

$$= \frac{1}{\sqrt{30}}\{0f(0)+1f(1)+3f(2)+4f(3)+2f(4)+0f(5)+...+0f(31)\}$$

$$= \frac{1}{\sqrt{30}}\{1f(1)+3f(2)+4f(3)+2f(4)\}$$

It is clear that for $j=1$ the window moves with step-size equals 1 and two elements are produced in each window placing. Consequently, the signal size is magnified to double. In the same way, one can see for any value of j that

(i) the window size is decreased by a factor of $1/2^j$,

(ii) the window moves with step-size equals 1,

(iii) 2^j elements are produced in each window placing, hence the signal size is magnified by a factor of 2^j.

Inversion:

If T denotes the transform matrix consisting of the family of wavelets for $j=1$, then we have

$$T = \frac{1}{\sqrt{30}} \begin{bmatrix} 2 & 4 & 3 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 3 & 4 & 2 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 2 & 4 & 3 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 3 & 4 & 2 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 2 & 4 & 3 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 3 & 4 & 2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 2 & 4 & 3 & 1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 2 & 4 & 3 & 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 3 & 4 & 1 \end{bmatrix}$$

After ignoring first three columns of T, we can write T^{1} as given below:

$$T^{1} = \frac{1}{\sqrt{60}} \begin{bmatrix} 1 & 2 & 3 & 4 & 4 & 3 & 2 & 1 & 0 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & 1 & 2 & 3 & 4 & 4 & 3 & 2 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 2 & 3 & 4 & 4 & 3 & 2 & 1 & 0 & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 2 & 3 & 4 & 4 & 3 & 2 & 1 \end{bmatrix}$$

One can see that $(T')'T' = TT' = I$ provided that T be treated as a family of orthonormal wavelets in the sense described in Section(7.2). Since $TT' = I \Rightarrow T' = T^{-1}$, therefore for two functions F and C, if $TF = C$ be called a Wavelet Transform then $T^{-1}C = F$ will be called Inverse Wavelet Transform (and vice versa) in the sense of data compression.

Software and Description of Automatic Nature of Zoom-out CWT

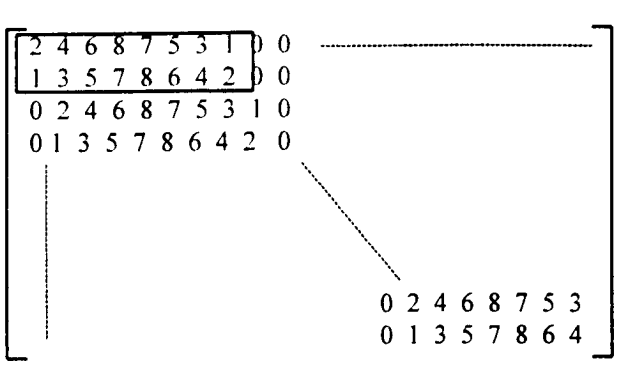
Software 7.4: In order to demonstrate the automatic wavelet compressing zoom-out wavelet transform, a program **tawczot.cpp** has been developed and listed in Appendix(2.1) and its outputs are shown in Figure(7.4.1).

This software is important to demonstrate a magical behaviour, of a zoom-out and noise reducing wavelet transform, of decreasing the window width on the increase of data

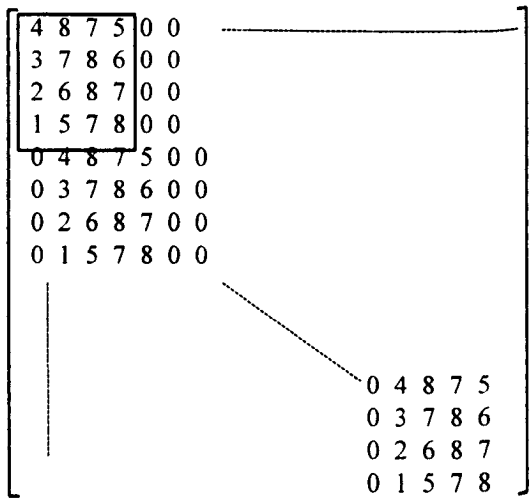
expansion. automatically. This behaviour is blessing to watch the higher frequency data segments in more details, and conversely, as compared to that having lower frequency. Although the fact of automatic variation of wavelet-width can be observed from the listing of the program and its outputs, yet it is interesting to present another example that follows: Consider, for example the wavelet

$$\frac{1}{\lambda}\{\psi(2^0 i)\}_{i\in Z} = \frac{1}{\sqrt{408}}\{\ ...,0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 8,\ 7,\ 6,\ 5,\ 4,\ 3,\ 2,\ 1,\ 0,...\ \}$$

consisting of sixteen non-zero elements as a basic wavelet. A large matrix is constructed by writing the non-zero-elements at its first column having all the leading elements zero. In order to expand data to ½ resolution (or otherwise ¼ resolution) we write the same non-zero elements on the 2nd column preceded by two zero entries (four zero entries for ¼ resolution) and all the leading elements are zero. Going on in this way, at least nine (six for ¼ resolution) columns are written to get a meaningful stage at which we ignore first fourteen (twelve for ¼ resolution) rows to have the shape of the matrix as given below:



(A) The wavelets produced automatically by the IWT from basic wavelet when we require the expansion of data to ½ resolution.



(B) The wavelets produced automatically by the IWT from the basic wavelet when we require the output of ¼ resolution.

The fact that "the larger the value of j the larger will be the magnification and smaller the noise reduction" can be seen in the above matrices A and B by observing their identical couples and 4-tuples, respectively. The step size of each two wavelets in A is 2

while that of each of the four wavelets in B is 4: hence the difference in resolution. On the other hand, the width of each wavelet in A is 8 while that of in B is 4; hence the difference in smoothing. Thus the case of B, in comparison, is suitable if we need to observe higher frequency contents by expanding it without spoiling (through smoothing) the small amplitudes.

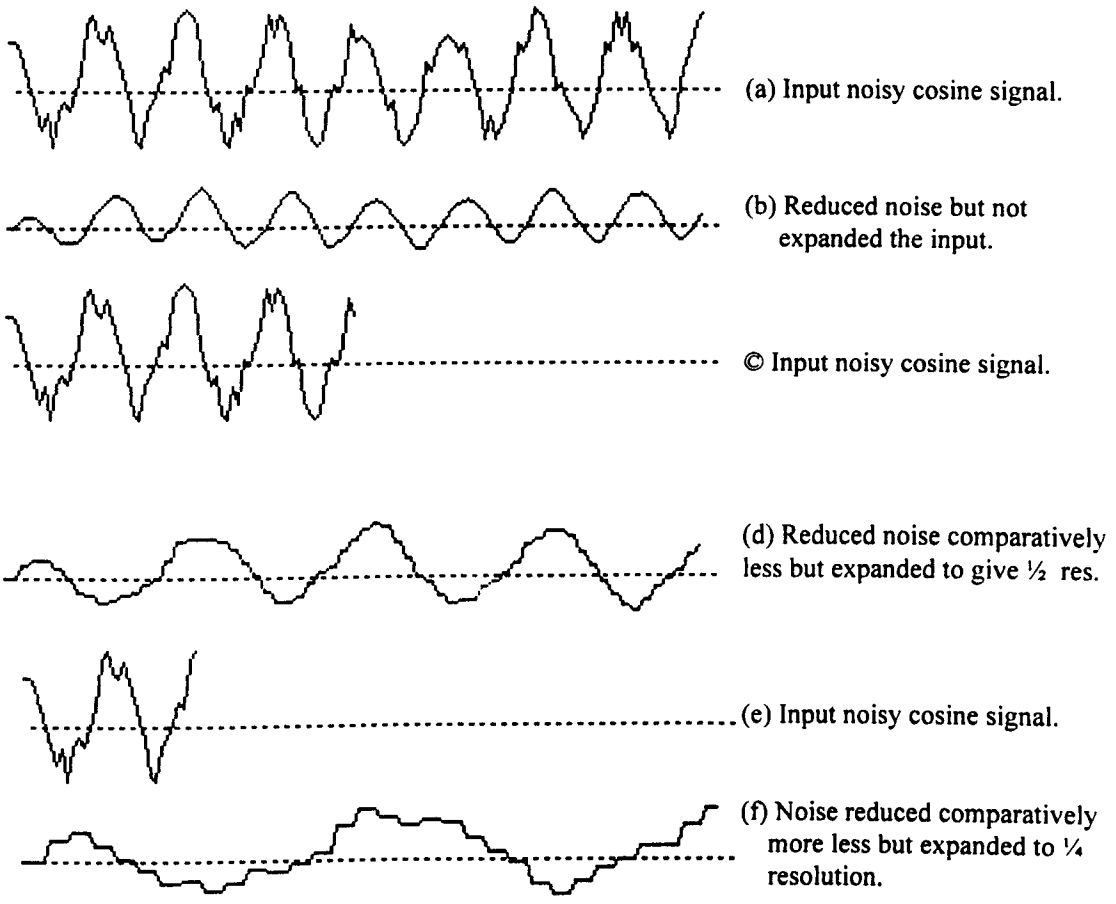


Figure 7.4.1: Outputs of tawczot.cpp to demonstrate automatic wavelet compressing zoom-out wavelet transform.

7.5 Automatic Wavelet Expansion and Data Compression (multiresolution zoom-in DWT)

The drawback free style of defining wavelets for zoom-in plus noise reduction wavelet transform is expressed in Section(7.2) and is given in Relation(7.2.3) and in Relation(7.2.4). Using this style with m=4 and j=0 in Relation(7.2.4), we have the wavelet

$$\frac{1}{\lambda}\{\psi(2^{0}i)\}_{i\in Z} = \frac{1}{\sqrt{10}}\{..., 0, 1, 2, 2, 1, 0, ...\}$$

which will be used as the basic wavelet in showing automatic wavelet expansion and data compression plus noise reduction. Using this wavelet basically and setting j=-1 in Equations(7.1.4), we have

$$f_{-1}(0) = \frac{\sqrt{2}}{\sqrt{10}}\{f(0)\psi(1/2) + f(1)\psi(1) + f(2)\psi(3/2) + f(3)\psi(2) + f(4)\psi(5/2) + f(5)\psi(3) + f(6)\psi(7/2) + f(7)\psi(4)+...+f(31)\psi(16)\}$$

$$= \frac{1}{\sqrt{5}}\{(1/2)f(0) + 1f(1) + (3/2)f(2) + 2f(3) + 2f(4) + (3/2)f(5) + 1f(6) + (1/2)f(7) + 0f(8)+...+0f(31)\}$$

$$= \frac{\sqrt{3}}{\sqrt{60}}\{1f(0) + 2f(1) + 3f(2) + 4f(3) + 4f(4) + 3f(5) + 2f(6) + 1f(7)\}$$

$$f_{-1}(1) = \frac{\sqrt{2}}{\sqrt{10}}\{f(0)\psi(-1/2) + f(1)\psi(0) + f(2)\psi(1/2) + f(3)\psi(1) + f(4)\psi(3/2)+...+ f(7)\psi(3) + f(8)\psi(7/2) + f(9)\psi(4)+...+f(31)\psi(15)\}$$

$$= \frac{1}{\sqrt{5}}\{0f(0) + 0f(1) + (1/2)f(2) + 1f(3) + (3/2)f(4) + 2f(5) + 2f(6) (3/2)f(7) + 1f(8) + (1/2)f(9) + 4f(10) + 0f(10)+...+0f(31)\}$$

$$= \frac{\sqrt{3}}{\sqrt{60}}\{1f(2) + 2f(3) + 3f(4) + 4f(5) + 4f(6) + 3f(7) + 2f(8) + 1f(9)\}$$

It is clear that for negative values of j

(i) the original window (in this case $\frac{1}{\sqrt{10}}\{..., 0, 1, 2, 2, 1, 0, ...\}$) expands 2^{-j} times (in this case equals $\frac{1}{\sqrt{60}}\{1,2,3,4,4,3,2,1\}$)

(ii) the step-size of the widow equals 2^{-j} (in this case 2)

(iii) and the transform achieved has an energy balancing factor (in this case $\sqrt{3}$).

Inversion: If T denotes the transform matrix consisting of the family of wavelets for j=-1, then we have

$$T = \begin{bmatrix} 1\ 2\ 3\ 4\ 4\ 3\ 2\ 1\ 0\ 0\ 0 \ldots\ldots\ldots\ldots\ldots\ldots 0 \\ 0\ 0\ 1\ 2\ 3\ 4\ 4\ 3\ 2\ 1\ 0 \ldots\ldots\ldots\ldots\ldots\ldots 0 \\ 0\ 0\ 0\ 0\ 1\ 2\ 3\ 4\ 4\ 3\ 2\ 1\ 0 \ldots\ldots\ldots\ldots\ldots 0 \\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 2\ 3\ 4\ 4\ 3\ 2\ 1\ 0 \ldots\ldots\ldots\ldots 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ 0 \ldots\ldots\ldots\ldots\ldots\ldots 0\ 1\ 2\ 3\ 4\ 4\ 3\ 2\ 1\ 0\ 0 \\ 0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 0\ 1\ 2\ 3\ 4\ 4\ 3\ 2\ 1 \end{bmatrix}$$

After ignoring the first 5 columns we can take transpose T^t which is exactly equals T given in Section(7.4) and since $T^t = T^{-1}$, therefore the two transforms given in Section(7.4) and in this Section(7.5) are Inverse Wavelet Transforms of each other.

Software 7.5: In order to demonstrate the automatic wavelet expanding zoom-in wavelet transform, a program tawezit.cpp has been developed and listed in Appendix(2.2) and its outputs are shown in Figure(7.5.1).

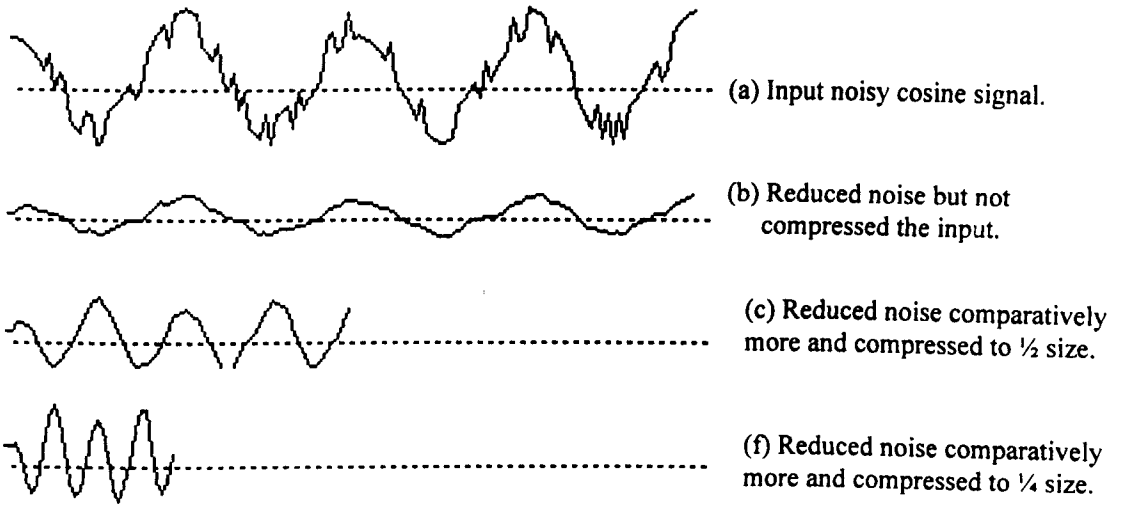


Figure 7.5.1: Outputs of tawezit.cpp to demonstrate automatic wavelet expanding zoom-in wavelet transform.

7.6 A Multi-Option ANN for Noise Reduction and Data Compression

In practical problems we need to process a variety of signals and images for only the noise reduction or both the noise reduction and the data compression simultaneously. The data to noise ratio can not be forced to be a constant in different signals and images. In addition, we need to compress data sometimes less and sometimes more. Moreover, we want to avoid a feedback ANN. All these objective can be achieved by designing a multi-option ANN which should be able to select a smaller or a larger window depending upon the amount of noise and/or our required level of data compression.

In co-ordination with the desired ANN, there should be a noise detector ANN that can suggest a smaller or a larger wavelet depending upon the amount of noise. The phase detector ANN presented in Section(10.2.4) of Chapter10 can be used as a noise detector or an ANN to detect noise/signal ratio can be designed on the logic of the software program **tremann.cpp** presented in Section(5.9). However, the desired ANN can be instructed by programming a soft switch or installing a hard switch.

Consider for examples three wavelets shown in Figure(7.6.1) which can be inherited in each output neurone of the ANN shown in Figure(7.6.2) so that it can use the elements of a selected wavelet as weights on its input connections. Since the number of elements in each of the three wavelets is 16, therefore we connect 16 inputs to each output neurone as shown in Figure(7.6.2).

$$\frac{1}{\sqrt{408}}\begin{bmatrix}1\\2\\3\\4\\5\\6\\7\\8\\8\\7\\6\\5\\4\\3\\2\\1\end{bmatrix} \qquad \frac{1}{\sqrt{60}}\begin{bmatrix}1\\2\\3\\4\\4\\3\\2\\1\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \qquad \frac{1}{\sqrt{10}}\begin{bmatrix}1\\2\\2\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

Figure 7.6.1: Three wavelets to be stored in each output neuron

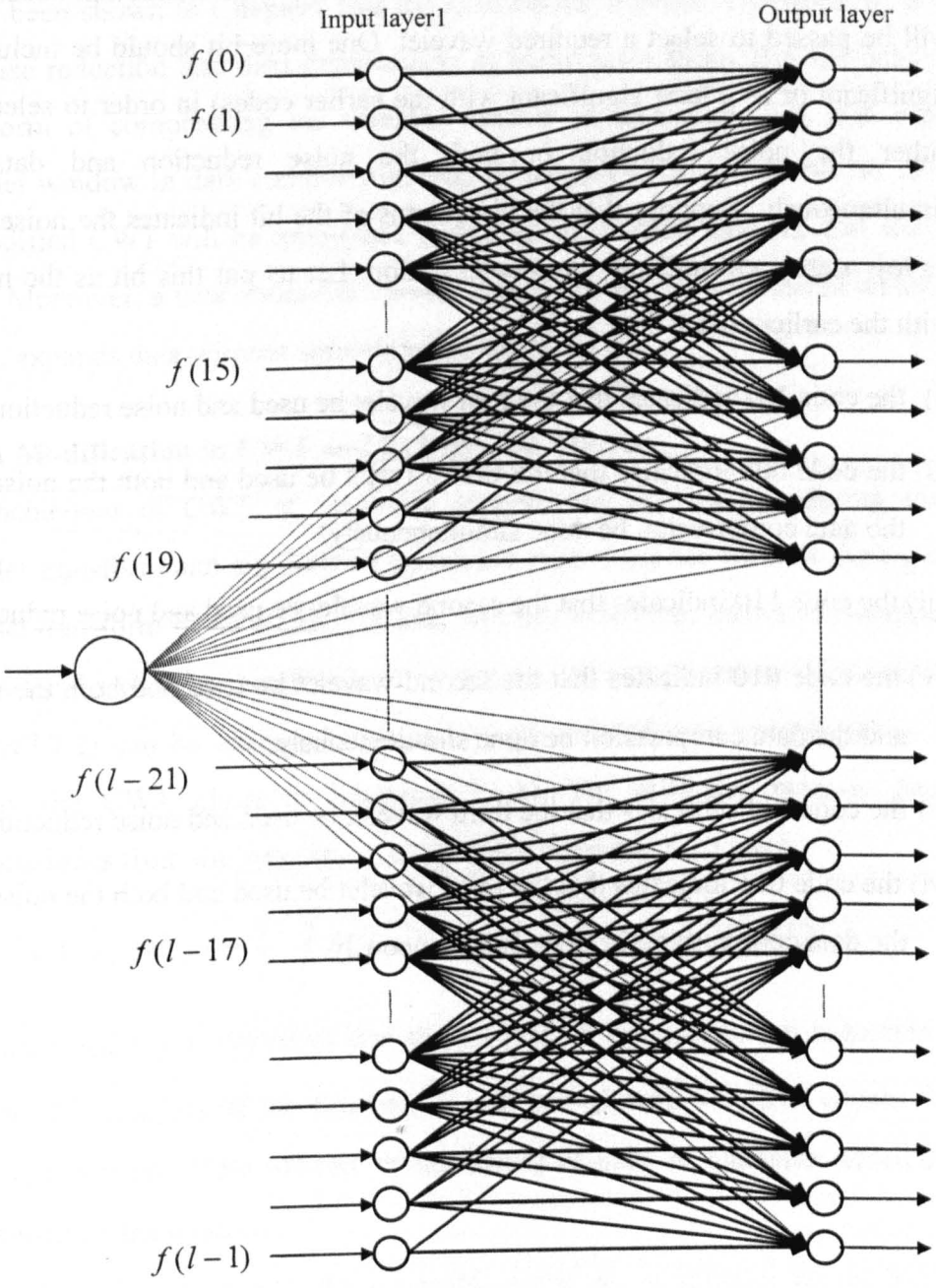


Figure 7.6.2: A multi-Option ANN for Noise Reduction and Data Compression

It can be seen in Figure(7.6.2) that there are 2 input layers. The first input layers consists of 1 neurone which passes a 3-bit instruction code to each output neurone.

Since there are three wavelets, therefore one of the three instruction codes 01, 10 and 11 will be passed to select a required wavelet. One more bit should be included (as a most significant or as a least significant with the earlier codes) in order to select the option of either the noise reduction or both the noise reduction and data compression simultaneously. Suppose that the high status of the bit indicates the noise reduction then its low status will indicate the second option. Let us put this bit as the most significant with the earlier codes, then

(i) the code 101 indicates that the first wavelet be used and noise reduction be done,

(ii) the code 001 indicates that the first wavelet be used and both the noise reduction and the data compression be done simultaneously,

(iii) the code 110 indicates that the second wavelet be used and noise reduction be done,

(iv) the code 010 indicates that the second wavelet be used and both the noise reduction and the data compression be done simultaneously,

(v) the code 111 indicates that the third wavelet be used and noise reduction be done,

(vi) the code 011 indicates that the third wavelet be used and both the noise reduction and the data compression be done simultaneously.

Continuous Wavelet Transform and its Modification

It has been shown in Chapter7 that the Continuous Wavelet Transform (CWT) provides us noise reduction and data compression at many resolutions. But the behaviour of the transform of compressing the wavelet window in data expansion and expanding the wavelet window in data compression may be tedious to design ANNs for the transform. A modified CWT will be introduced in this chapter which uses wavelet windows of fix sizes. Moreover, a new zoom-out wavelet transform will be introduced which, unlike the CWT, expands data without smoothing the original data.

8.1 A Modification in CWT and its Digitised Version

The behaviour of CWT of automatic compressing the wavelet during the zoom-out wavelet transform and similarly of automatic expanding the wavelet during the zoom-in wavelet transform is, no doubt, blessing. But this behaviour adds complications to design ANNs for the wavelet transforms. For example, the complications of the ANN shown in Figure(7.7.2) can be observed. In order to get ride of such complications we need to modify the CWT given in Equations(7.1.4). We shift the zoom-in and zoom-out characteristics from the wavelet ψ to the input function f to have

$$CWT_f(k,j) = \frac{1}{\lambda}\int_0^{m-1} f(x+2^j k)\psi(x)dx\,,\quad k = 0,1,...,l/2^j\,. \tag{8.1.1}$$

This modified CWT expresses that the $(l-m)/2^j+1$ successive segments overlapped by $m-2^j$ elements of the function f are passed with step-size equals 2^j for noise reduction in front of the wavelet ψ consisting of only m elements which are non-zero and constitute the window.

Digitisation: If we digitise the functions f and ψ at integral values and denote the wavelet transform of the function f by $f_j(k)$, then from Equations(8.1.1) we have

$$f_j(k) = \frac{1}{\lambda}\sum_{i=0}^{m-1} f(i+2^j k)\psi(i)\,,\text{ where } k = 0,1,...,l/2^j \tag{8.1.2}$$

The performance of the transform for noise reduction and data compression are checked against the self developed program **tnrzi.cpp** which is listed in Appendix(3.3) and its outputs are shown in Figure(8.3.2).

Digitised signals and images consist of values (elements) rather than the definitions of their generating functions. But the application of the transform given in Equations(8.1.2) for data expansion which corresponds to the negative values of j, the presence of the term $k/2^j$ in $f(i+k/2^j)$, $j>0$ demands us unavailable values to insert between each two consecutive elements of the signal f. In other words, the signal f is treated as it were a continuous one. An easy fix to this problem lies in dropping the factor $1/2^j$ from $f(i+k/2^j)$ to have $f(i+k)$. Its performance is good and has been checked against the self developed program **tnrzo.cpp**, listed in Appendix(3.1), whose outputs are shown in Figure(8.1.1). However, in order to remove this drawback and to have a new zoom-out wavelet transform that can expand data without smoothing the original data, we can further modify the transform given in Equation(8.1.2).

Further Modification:

If $Q(\frac{k}{2^j})$ and $R(\frac{k}{2^j})$ denote the quotient and remainder after division of k by 2^j, then the unavailable values of f at non integral indices can approximately be constructed as:

$$f(i+\frac{k}{2^j}) \cong \frac{1}{2^j}\begin{cases}(2^j-1)f(i+Q(\frac{k}{2^j}))+f(i+Q(\frac{k}{2^j})+1), & 0<R(\frac{k}{2^j})\le\frac{1}{2}\\ f(i+Q(\frac{k}{2^j}))+(2^j-1)f(i+Q(\frac{k}{2^j})+1), & \frac{1}{2}<R(\frac{k}{2^j})<1\end{cases} \qquad (8.1.3)$$

The computed values are more influenced by closer element and less by the other. The more the closeness the more is the influence. See Section(8.4) for its implementation. At the moment, it seems complicated and time consuming to implement this modification. But it should be remember that we are not going to implement it as it is. Instead, we will make it unbelievably interesting and efficient in Section(8.6).

Software 8.1: To check Equations(8.1.4), the program **tnrzo.cpp** is developed, listed in

$$f_j(ak+r)=\frac{1}{\lambda}\sum_{i=0}^{m-1} f(i+k)\psi(i);\ \ a=2^j,\ \ r=1,2,...a,\ \ k=0,1,...,l/2^j \qquad (8.1.4)$$

Appendix(3.1). and its outputs are shown in Figure(8.1.1).

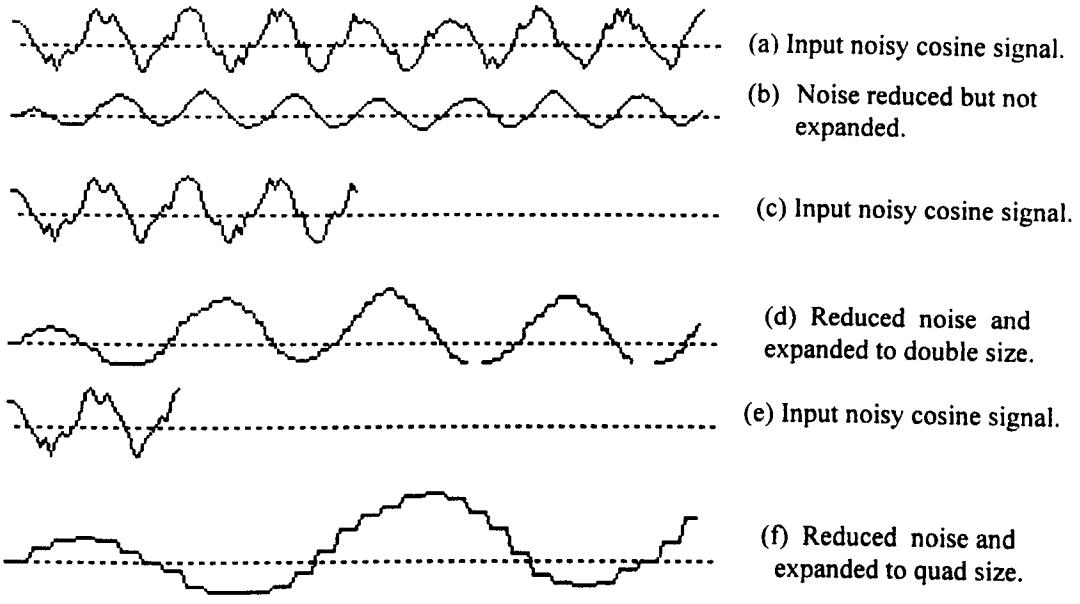


Figure 8.1.1: Outputs of tnrzo.cpp to demonstrate noise reducing and data expanding wavelet transform.

8.2 A Noise Reducing ANN for CWT

When the value of resolution parameter j is set equals 0, the system of Equations(8.1.2) provides us a noise reducing ANN and keeps the size of the signal f the same. By putting j=0 in Equations(8.1.2), we have

$$f_0(k) = \frac{1}{\lambda}\sum_{i=0}^{m-1} f(i+k)\psi(i), \text{ where } k = 0,1,...,l-m. \tag{8.2.1}$$

This system of equations can equivalently be represented in the matrix form as given below:

$$\frac{1}{\lambda}\begin{bmatrix} f(0) & f(1) & \cdots & f(m-1) \\ f(1) & f(2) & \cdots & f(m) \\ \cdots & \cdots & \cdots & \cdots \\ f(l-m) & f(l-m+1) & \cdots & f(l-1) \end{bmatrix}\begin{bmatrix} \psi(0) \\ \psi(1) \\ \cdot \\ \psi(m-1) \end{bmatrix} = \begin{bmatrix} f_0(0) \\ f_0(1) \\ \cdot \\ f_0(l-m) \end{bmatrix}$$

We design an ANN as given in Figure(8.2.1). Let us set m=8, then the window values $\psi(0), \psi(1), \ldots, \psi(7)$ will act as the weights on input connections of each output neurone of the ANN. The zero-padding technique has been used there to compute first four and the last three (total $2^j m-1$) outputs in order to make the output signal equal in size to the input signal f. Greater the value of m, more the reduction of noise.

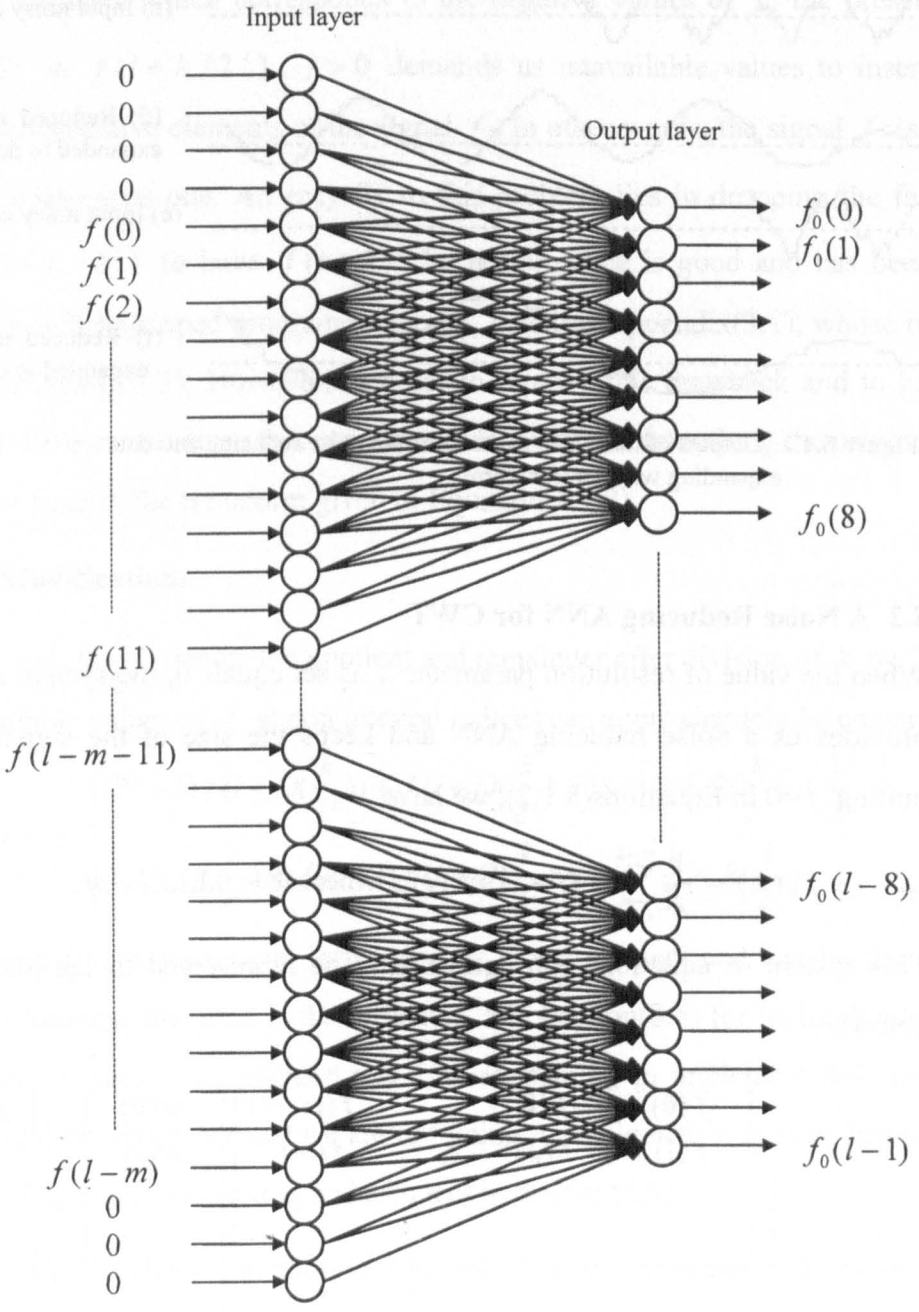
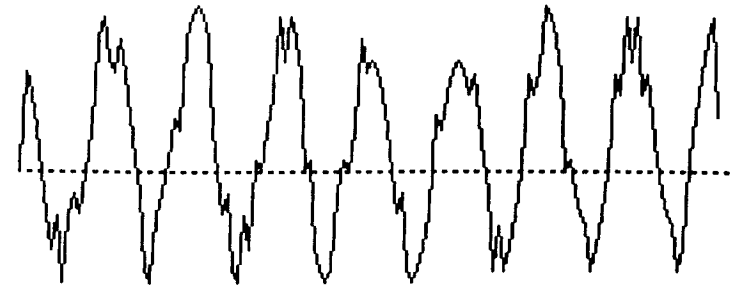
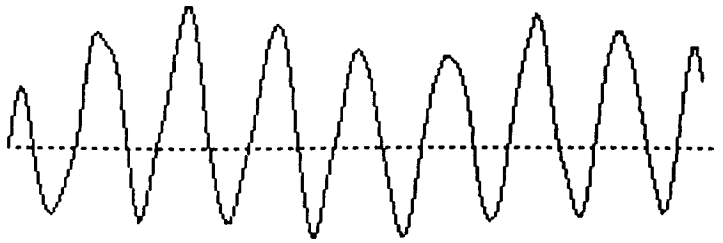


Figure 8.2.1: A Noise Reducing ANN for CWT

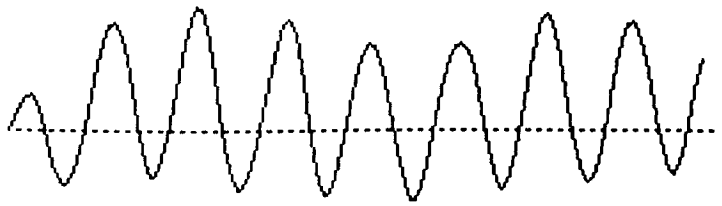
Software 8.2: Although the code for nose reduction is included in **tnrzo.cpp** and **tnrzi.cpp**. yet a separate program **tnr.cpp** (testing of noise reduction) is developed, listed in Appendix(3.2), and its outputs are shown in Figure(8.2.2).



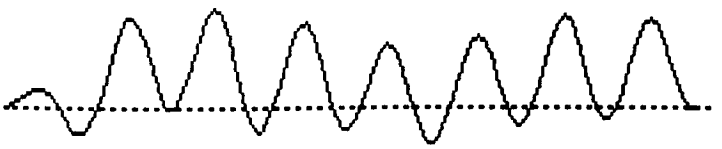
(a) Input Noisy cosine signal.



(b) Output when width of wavelet is 4.



(c) Output when width of wavelet is 8.



(d) Output when width of wavelet is 16.

Figure 8.2.2: Outputs of tnr.cpp to demonstrate noise reducing wavelet transform.

8.3 A Zoom-in and Noise Reducing ANN for CWT

In this Section we will see that when j is positive the system of Equations(8.1.2) performs zoom-in and noise reduction simultaneously. Putting $j = 1$ in Equations(8.1.2), we have

$$f_1(k) = \frac{1}{\lambda}\sum_{i=0}^{m-1} f(i+2k)\psi(i), \text{ where } k = 0,1,....,l-m. \qquad (8.3.1)$$

The system of Equations(8.3.1) can equivalently be represented in the matrix form as given below:

$$\frac{1}{\lambda}\begin{bmatrix} f(0) & f(1) & \cdots & f(m-1) \\ f(2) & f(3) & \cdots & f(m+1) \\ f(4) & f(5) & \cdots & f(m+3) \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ f(l-m) & f(l-m+1) & \cdots & f(l-1) \end{bmatrix}\begin{bmatrix} \psi(0) \\ \psi(1) \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \psi(m-1) \end{bmatrix} = \begin{bmatrix} f_1(0) \\ f_1(1) \\ \cdot \\ \cdot \\ \cdot \\ f_1(\frac{l-m}{2}-1) \end{bmatrix} \tag{8.3.2}$$

Putting $j = 2$ in Equations(8.1.2), we have

$$f_2(k) = \frac{1}{\lambda}\sum_{i=0}^{m-1} f(i+4k)\psi(i), \text{ where } k = 0,1,\ldots,l-m. \tag{8.3.3}$$

The system of Equations(8.3.3) can equivalently be represented in the matrix form as given below:

$$\frac{1}{\lambda}\begin{bmatrix} f(0) & f(1) & \cdots & f(m-1) \\ f(4) & f(5) & \cdots & f(m+3) \\ f(8) & f(9) & \cdots & f(m+5) \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ f(l-m) & f(l-m+1) & \cdots & f(l-1) \end{bmatrix}\begin{bmatrix} \psi(0) \\ \psi(1) \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \psi(m-1) \end{bmatrix} = \begin{bmatrix} f_2(0) \\ f_2(1) \\ \cdot \\ \cdot \\ \cdot \\ f_2(\frac{l-m}{4}-1) \end{bmatrix}$$

This shows that larger j implies larger step implies more compression.

To perform the matrix multiplication given in Relation(8.3.2) we design an ANN as given in Figure(8.3.1). Let us set m=4, then the window values $\psi(0), \psi(1), \psi(2), \psi(3)$ will act as weights on the input connections of each output neurone of the ANN. The zero-

padding technique has been used here to compute first (total $2^{-j}m-1$) output in order to make the output signal half in size to the input signal f.

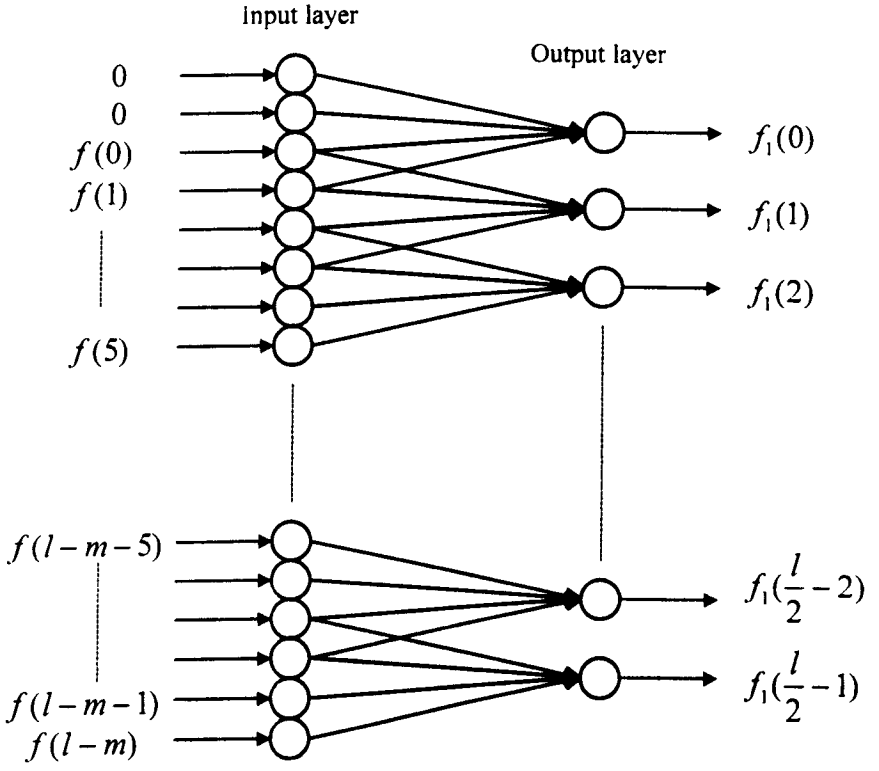


Figure 8.3.1: A Zoom-in and Noise Reducing ANN for CWT

For noise reduction the behaviour of the system given in Equations(8.3.2) depends upon the window width such that larger window implies more reduction of noise. This fact can also be seen in Chapter9 where the performance of some one-dimensional and two-dimensional wavelets will be tested through C++ coded programs.

Software 8.3: In order to demonstrate the data compressing and noise reducing wavelet transform which uses a single wavelet and keeps its size the fixed, a program **tnrzi.cpp** has been developed and listed in Appendix(3.3) and its outputs are shown in Figure(8.3.2).

(a) Input noisy cosine signal.

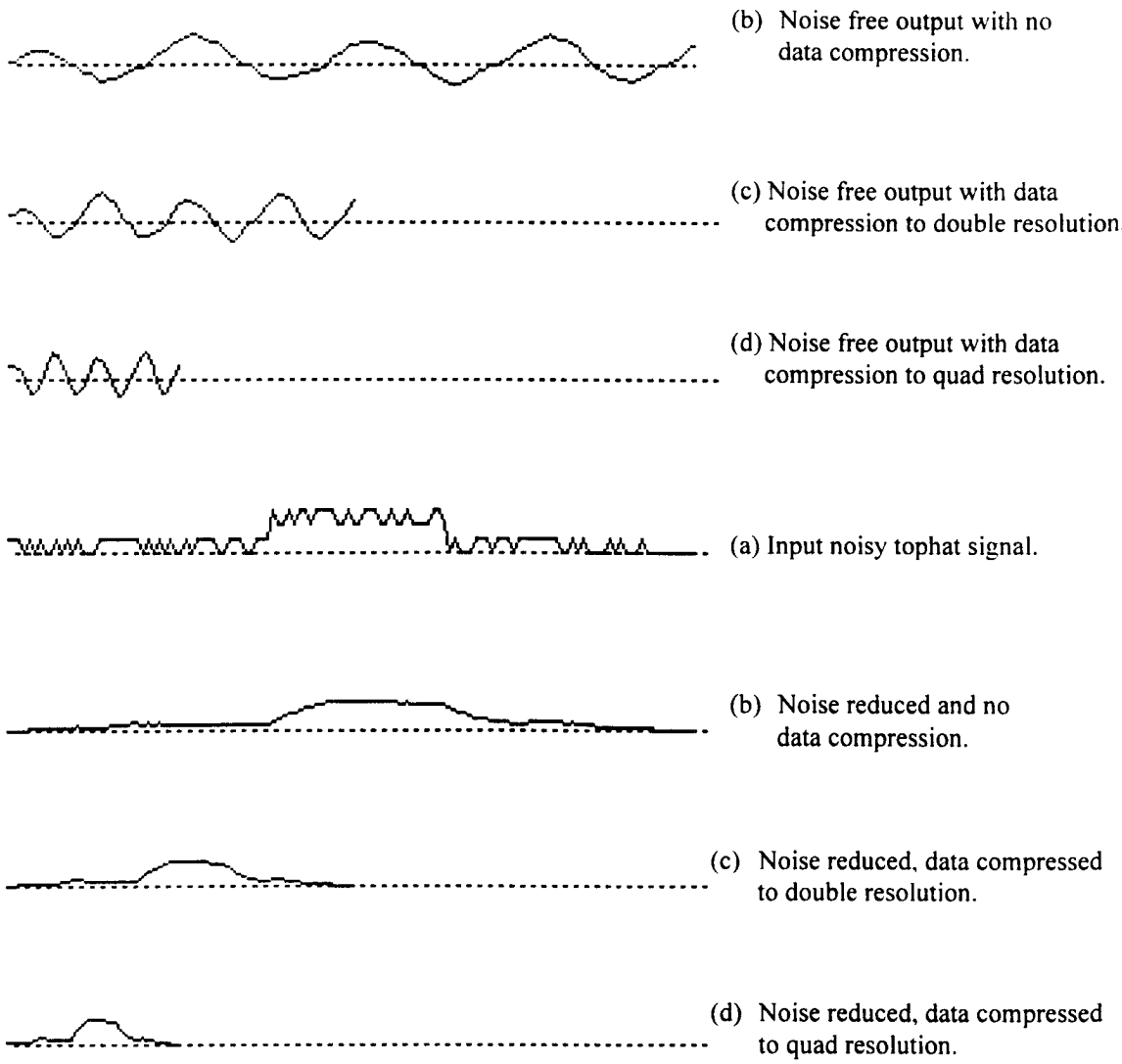


Figure 8.3.2: Outputs of tnrzi.cpp to demonstrate noise reducing and data compressing wavelet transform.

8.4 A Zoom-out and Noise Reducing ANN for CWT

It is clear from Section(8.2) that when the resolution parameter $j=0$, the system of Equations(8.1.2) provides us simply a noise reducing CWT keeping the resolution of the input and output signals the same. In this Section we will see that when j is negative the system of Equations(8.1.2) performs zoom-out and noise reduction simultaneously. Putting $j=-1$ in Equations(8.1.2), we have

$$f_1(k) = \frac{1}{\lambda}\sum_{i=0}^{m-1} f(i+\frac{k}{2^1})\psi(i)\,,\ \text{where } k = 0.1,.....,l-m. \qquad (8.4.1)$$

The system of Equations(8.4.1) can equivalently be represented in the matrix form as given below:

$$\frac{1}{\lambda}\begin{bmatrix} f(0) & f(1) & \cdots & f(m-1) \\ f(\frac{1}{2}) & f(\frac{3}{2}) & \cdots & f(m-\frac{1}{2}) \\ f(1) & f(2) & \cdots & f(m) \\ f(\frac{3}{2}) & f(\frac{5}{2}) & \cdots & f(\mathrm{m}+\frac{1}{2}) \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ f(l-m-1) & f(l-m) & \cdots & f(l-2) \\ f(\mathrm{l}-m-\frac{1}{2}) & f(l-m+\frac{1}{2}) & \cdots & f(l-\frac{3}{2}) \\ f(l-m) & f(l-m+1) & \cdots & f(l-1) \end{bmatrix}\begin{bmatrix} \psi(0) \\ \psi(1) \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \psi(m-1) \end{bmatrix} = \begin{bmatrix} f_1(0) \\ f_1(1) \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ f_1(2l-1) \end{bmatrix} \qquad (8.4.2)$$

The values of f at non integral indices should be computed by using Equation(8.1.3). Putting $j = -2$ in Equations(8.1.2), we have

$$f_2(k) = \frac{1}{\lambda}\sum_{i=0}^{m-1} f(i+\frac{k}{2^2})\psi(i)\,,\ \text{where } k = 0,1,....,l-m. \qquad (8.4.3)$$

One can represent the system of Equations(8.4.3) equivalently in the matrix form. It is clear from the systems given in Equations(8.4.1) and Equations(8.4.3) that larger j implies more magnification and lower resolution. For noise reduction, see Chapter(11).

To perform the matrix multiplication given in Relation(8.4.2) we design an ANN as given in Figure(8.4.1). Let us set m=4, then the window values $\psi(0),\psi(1),\psi(2),\psi(3)$ will act as weights on the input connections of each output neurone of the ANN. The last element (total $2^{-j}m-1$) is repeated in order to make the output signal double in size to the input signal f. The shaded neurones in Figure(8.4.1) apply Equation(8.1.3) to compute their outputs.

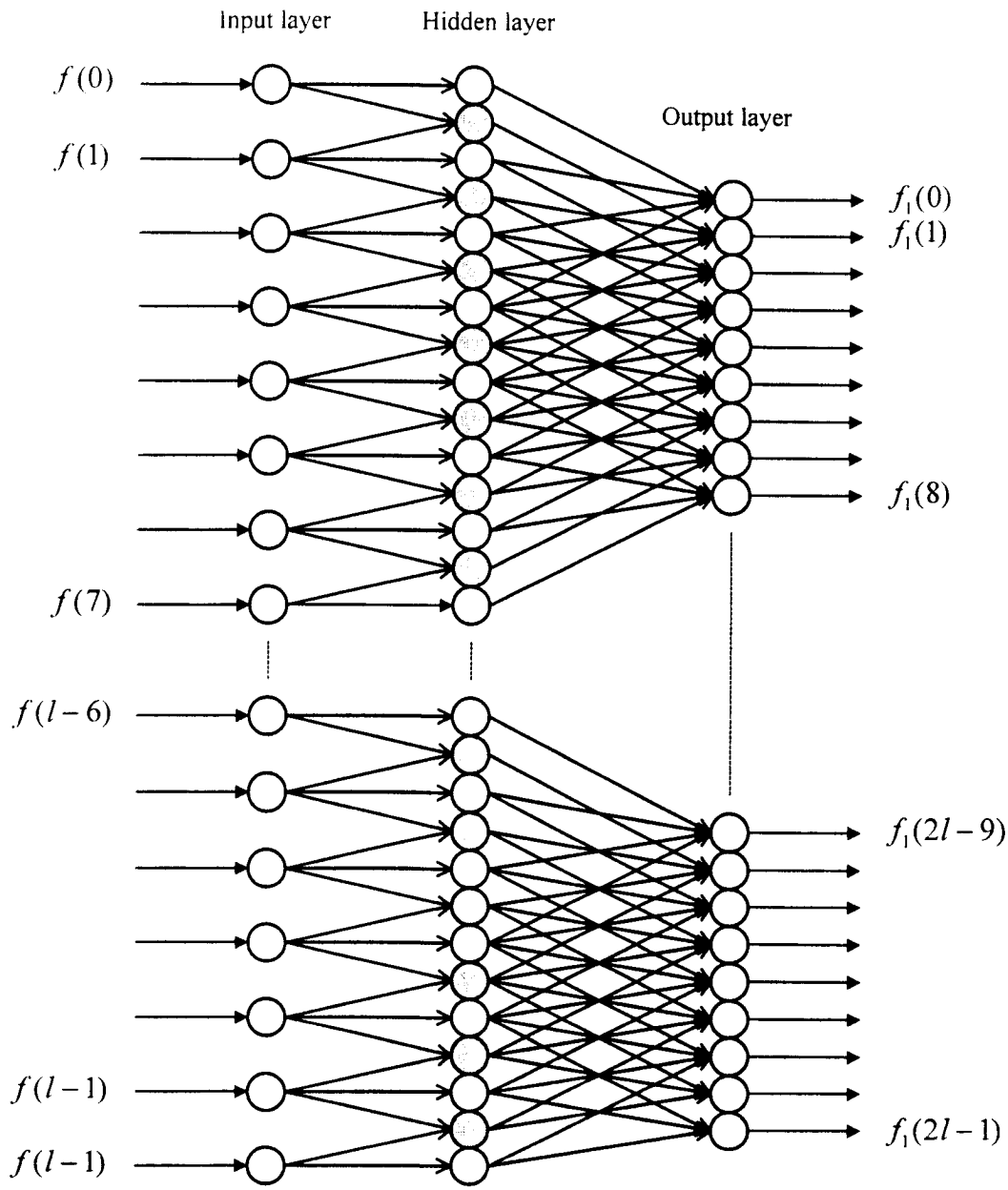


Figure 8.4.1: A Zoom-out and Noise Reducing ANN for CWT

8.5 An Inverse of Zoom-in CWT (a zoom-out CWT)

Let both A and B be column matrices and let T be a transform matrix consisting of an orthogonal family of wavelets such that $TA = B$. If T^{-1} exists, then the vector A is given

by $A = T^{-1}B$. This concept can be used to have a zoom-out CWT. We start from the system of equations for zoom-in CWT given in Section(8.4) and write it equivalently as

$$\frac{1}{\lambda}\begin{bmatrix} \psi_0 \psi_1 \psi_2 \dots \psi_{m-1} 0\ 0 \dots 0 \\ 0\ 0\ \psi_0 \psi_1 \psi_2 \dots \psi_{m-1} 0\ 0 \dots 0 \\ 0\ 0\ 0\ 0\ \psi_0 \psi_1 \psi_2 \dots \psi_{m-1} 0\ 0 \dots 0 \\ \dots \\ \dots \\ \dots \\ 0 \dots 0 \psi_0 \psi_1 \psi_2 \dots \psi_{m-1} 0\ 0 \\ 0 \dots 0\ 0\ \psi_0 \psi_1 \psi_2 \dots \psi_{m-1} \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ . \\ . \\ . \\ . \\ . \\ f(l-1) \end{bmatrix} = \begin{bmatrix} f_{-1}(0) \\ f_{-1}(1) \\ . \\ . \\ . \\ f_{-1}(\frac{l-m}{2}-1) \end{bmatrix} \qquad (8.5.1)$$

The matrix form given in Relation(8.5.1) can be denoted by $TA = B$. Note that the number of elements in each row of T is l out of which $l-m$ are non-zero. Apparently, this system has computing redundancy due to the existence of zero elements and can not be preferred in comparison of the system given in Section(8.3), however, the zero elements are merely place holders and should be ignored practically. Moreover, it is not the purpose here to present an alternative system for zoom-in CWT, instead we are interested in developing an algorithm for zoom-out CWT from the matrix form given in Relation(8.5.1).

If T^{-1} exists, then applying T^{-1} on both sides of $TA = B$, we have $T^{-1}TA = T^{-1}B \Rightarrow T^{-1}B = A$. There is no need to compute T^{-1} because $TT' = I \Rightarrow T^{-1} = T'$, so that

$$T'B = A. \qquad (8.6.2)$$

Relation(8.6.2) provides us the desired zoom-out CWT. For this transform, we design an ANN as given in Figure(8.6.1). Let us set m=4, then the window values $\psi(0), \psi(1), \psi(2), \psi(3)$ will act as weights of inputs of each output neurone. Zero-padding technique has been used to compute first four and the last three (total $2^j m - 1$) outputs in order to make the output signal double in size to the input signal f.

by $A = T^{-1}B$. This concept can be used to have a zoom-out CWT. We start from the system of equations for zoom-in CWT given in Section(8.3) and write it equivalently as

$$\frac{1}{\lambda}\begin{bmatrix} \psi_0 \psi_1 \psi_2 \dots \psi_{m-1} 0\ 0 \dots 0 \\ 0\ 0\ \psi_0 \psi_1 \psi_2 \dots \psi_{m-1} 0\ 0 \dots 0 \\ 0\ 0\ 0\ 0\ \psi_0 \psi_1 \psi_2 \dots \psi_{m-1} 0\ 0 \dots 0 \\ \dots \\ \dots \\ \dots \\ 0 \dots 0 \psi_0 \psi_1 \psi_2 \dots \psi_{m-1} 0\ 0 \\ 0 \dots 0\ 0 \psi_0 \psi_1 \psi_2 \dots \psi_{m-1} \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ . \\ . \\ . \\ . \\ . \\ f(l-1) \end{bmatrix} = \begin{bmatrix} f_{-1}(0) \\ f_{-1}(1) \\ . \\ . \\ . \\ f_{-1}(\frac{l-m}{2}-1) \end{bmatrix} \qquad (8.5.1)$$

The matrix form given in Relation(8.5.1) can be denoted by $TA = B$. Note that the number of elements in each row of T is l out of which $l-m$ are non-zero. Apparently, this system has computing redundancy due to the existence of zero elements and can not be preferred in comparison of the system given in Section(8.3), however, the zero elements are merely place holders and should be ignored practically. Moreover, it is not the purpose here to present an alternative system for zoom-in CWT, instead we are interested in developing an algorithm for zoom-out CWT from the matrix form given in Relation(8.5.1).

If T^{-1} exists, then applying T^{-1} on both sides of $TA = B$, we have $T^{-1}TA = T^{-1}B \Rightarrow T^{-1}B = A$. There is no need to compute T^{-1} because $TT' = I \Rightarrow T^{-1} = T'$, so that

$$T'B = A. \qquad (8.5.2)$$

Relation(8.5.2) provides us the desired zoom-out CWT. For this transform, we design an ANN as given in Figure(8.5.1). Let us set m=4, then the window values $\psi(0), \psi(1), \psi(2), \psi(3)$ will act as weights of inputs of each output neurone. Zero-padding technique has been used to compute first four and the last three (total $2^j m-1$) outputs in order to make the output signal double in size to the input signal f.

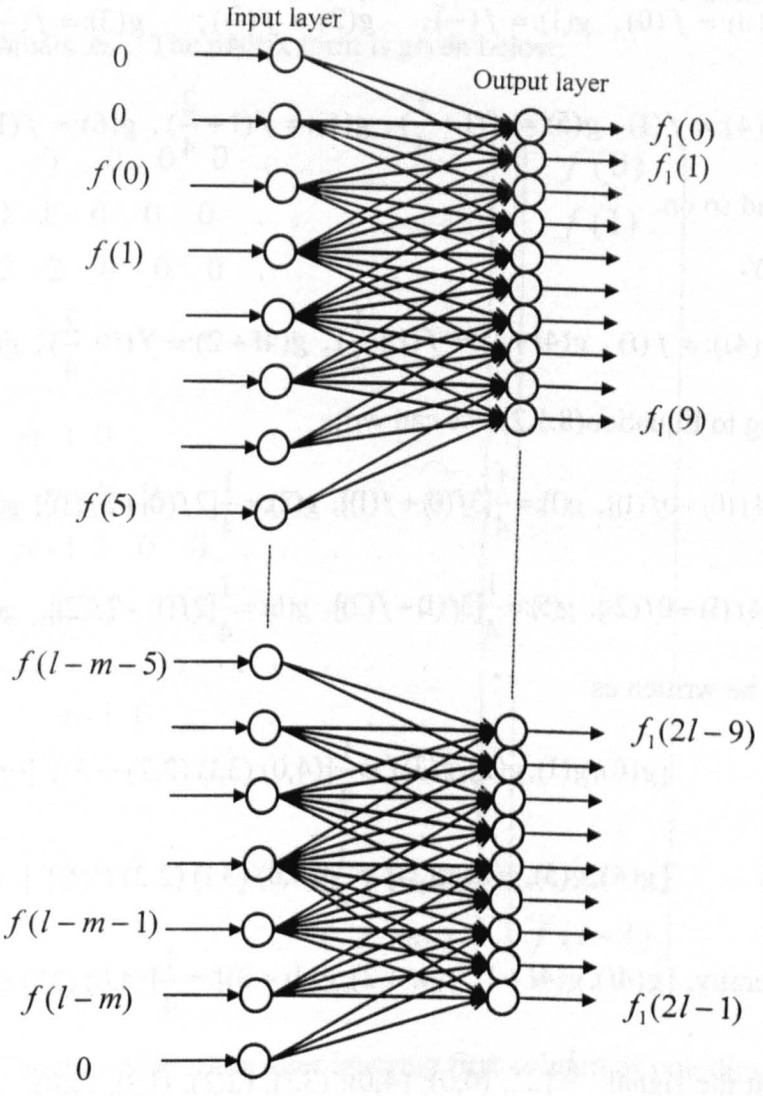


Figure 8.5.1: A Zoom-out and Noise Reducing ANN for CWT

8.6 A New Zoom-out CWT (Expanding data without reducing noise)

The systems given in Equations(8.4.1) can not be forced easily to expand data without reducing the noise. In this section a new algorithm for zoom-out CWT will be introduced which solves this problem.

A signal f can be expanded as much as we desire by using Equation(8.1.3). Let us consider the case when a digital signal $\{f(i)\}_{i=0}^{l-1}$ is expanded 4-times of its initial length. If $\{g(i)\}_{i=0}^{16}$ denotes the resulting signal then we can write

$$g(0):= f(0),\quad g(1):= f(\frac{1}{4}),\quad g(2):= f(\frac{2}{4}),\quad g(3):= f(\frac{3}{4}),$$

$$g(4):= f(1),\ g(5):= f(1+\frac{1}{4}),\ g(6):= f(1+\frac{2}{4}),\ g(6):= f(1+\frac{3}{4})$$

and so on.

Generally,

$$g(4i):= f(i),\ g(4i+1):= f(i+\frac{1}{4}),\ g(4i+2):= f(i+\frac{2}{4}),\ g(4i+3):= f(i+\frac{3}{4})$$

According to Equation(8.1.2), we can write

$$g(0):=\frac{1}{4}[4f(0)+0f(1)],\ g(1):=\frac{1}{4}[3f(0)+f(1)],\ g(2):=\frac{1}{4}[2f(0)+2f(1)],\ g(3):=\frac{1}{4}[f(0)+3f(1)],$$

$$g(4):=\frac{1}{4}[4f(1)+0f(2)],\ g(5):=\frac{1}{4}[3f(1)+f(2)],\ g(6):=\frac{1}{4}[2f(1)+2f(2)],\ g(6):=\frac{1}{4}[2f(1)+2f(2)],$$

This can be written as

$$[g(0),g(1),g(2),g(3)]^t=\frac{1}{4}[(4,0)\ (3,1)\ (2,2)\ (1,3)]^t\ [(f_0,f_1)],$$

$$[g(4),g(5),g(6),g(7)]^t=\frac{1}{4}[(4,0)\ (3,1)\ (2,2)\ (1,3)]^t\ [(f_1,f_2)],$$

and generally, $[g(4i),g(4i+1),g(4i+2),g(4i+3)]^t=\frac{1}{4}[(4,0)\ (3,1)\ (2,2)\ (1,3)]^t\ [(f_i,f_{i+1})]$.

Note that the signal $\frac{1}{4}\{....,\ (0,0),\ (4,0),\ (3,1),\ (2,2),\ (1,3),\ (0,0),\ ....\}$ can be said to be a *Double* ***Wavelet***. In general if a signal is required to expand m-times of its initial size, then we need the wavelet as given below:

$$\frac{1}{m}\{...,(0,0),(m-0,0),(m-1,1),...,(m-(\frac{m}{2}),(m-(\frac{m}{2}+1),m+1-(\frac{m}{2})),...,(1,m-1),(0,0),....\}$$

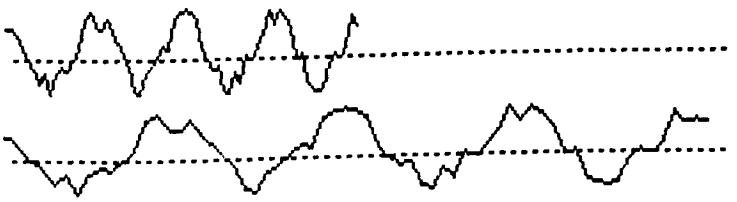
Interestingly, this phenomena can be expressed in the matrix form such that the window $\{0,1,...,m-1,m,m-1,...,1\}$ moves from column to column with step-size equals m. The energy of each row is also equals m. Unlike the zoom-out CWT given in Equation(8.4.1), here the required degree of zoom-out will depend upon the width of the window such that the signal will be expanded m time when step-size of the window of width $2m$ is m. The

first or the last element of the output signal can be repeated m-1 times to make the size of the signal equals ml. The matrix form is given below:

$$\frac{1}{m}\begin{bmatrix} m & 0 & 0 & 0 & 0 & \cdots & \cdots \\ m-1 & 1 & 0 & 0 & 0 & \cdots & \cdots \\ m-2 & 2 & 0 & 0 & 0 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & m-1 & 0 & \cdots & \cdots & \cdots & \cdots \\ 0 & m & 0 & 0 & \cdots & \cdots & \cdots \\ 0 & m-1 & 1 & 0 & 0 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 1 & m-1 & 0 & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & 0\ 2 & m-2 \\ \cdots & \cdots & \cdots & \cdots & \cdots & 0\ 1 & m-1 \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ f(l-1) \end{bmatrix} = \begin{bmatrix} g(0) \\ g(1) \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ g(ml-1) \end{bmatrix}$$

Inversion: The transpose taken after ignoring first column of one-dimensional transform matrix gives the Inverse Wavelet Transform.

Software 8.6: In order to demonstrate the development and the working of the wavelet transform which expand data without reducing its noise, a program **tzoomout.cpp** has been developed and listed in Appendix(3.3) and its outputs are shown in Figure(8.6).



(a) Noisy input cosine signal.

(b) Expanded at ½ resolution.

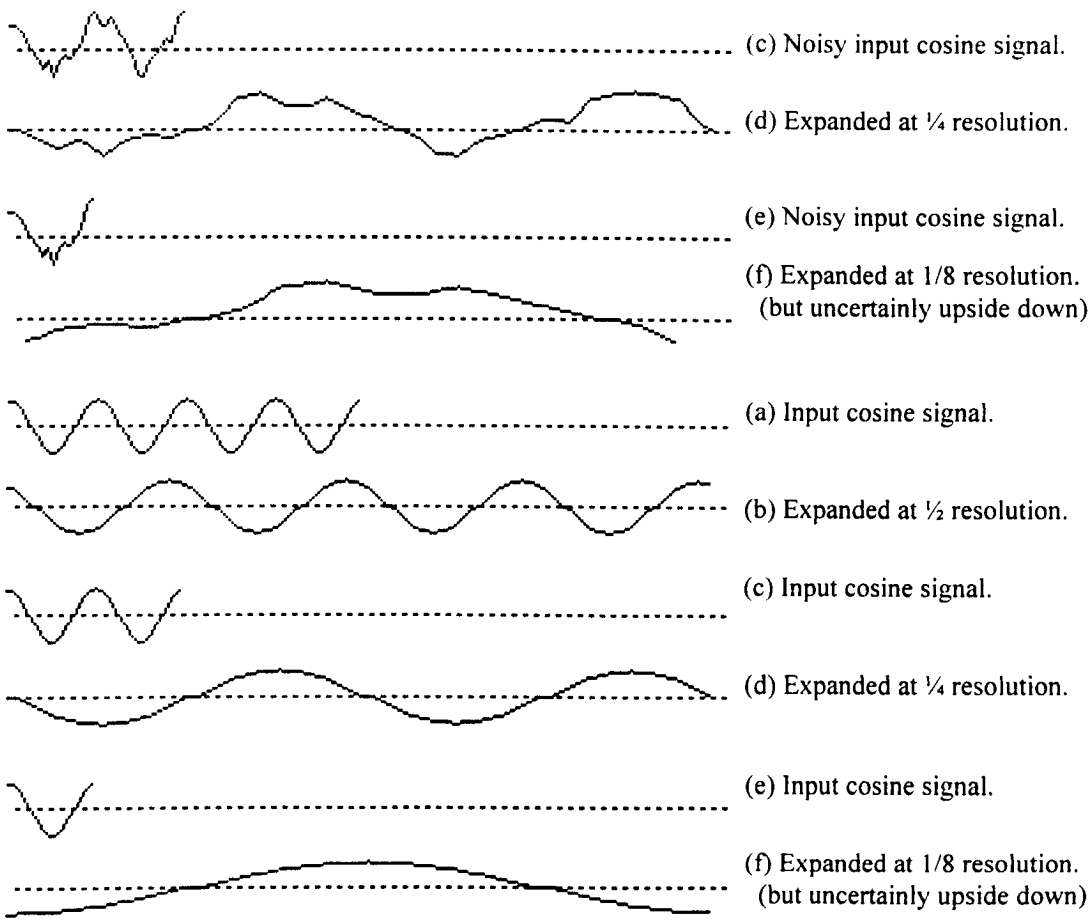


Figure 8.6: Outputs of tzoomout.cpp to demonstrate zooming-out without data compressing wavelet transform.

8.7 A Wavelet Noise Reducing Feedback ANN

Although noise can be reduced up to a pre-decided level in a single step by designing a feed forward ANN by choosing a suitable length of the window, but this ANN will not be versatile to process the signals having different amount of noise. So, we need a switch able feedback ANN based so that its repetitions can be stopped at any level. That is, we are interested here in reducing noise by using a small wavelet repeatedly.

As a first step, by using the formula given in Equations(8.1.2) we transform $f(k)$ keeping $j=0$ and get $c_0(k)$, $k=0,1,....,l-1$. In the second step , we feed back the values of $c_0(k)$ to the input lines of $f(k)$. That is, $f(k):=c_0(k)$ and repeat the first step. A soft or/and a hard switch can be installed in the ANN to put off the feedback lines

and put on the output lines when a required level of noise reduction is achieved. For m equals 4 a feedback ANN is given in Figure(8.7.1).

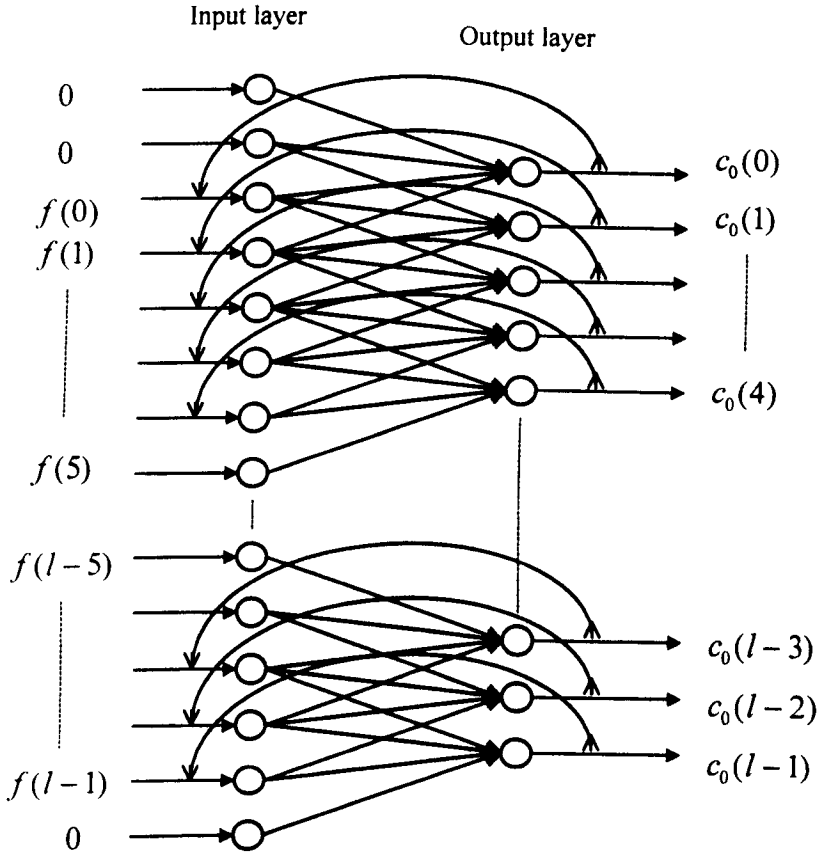


Figure 8.7.1: A wavelet noise reducing feedback ANN

8.8 A Wavelet Data Compressing Feedback ANN

Although data compression plus noise reduction can be done up to a pre-decided level in a single step by designing and using a feed forward ANN by choosing a suitable length of a suitable window, but this ANN will not be versatile to get different levels of data compression and noise reduction. So, we need a switch able feedback ANN so that its repetitions can be stopped at any level. In Equations(8.1.2), the window width m and the resolution parameter $j \leq -1$ should be such that

$$\text{step-size} = \frac{1}{2^j} \leq \frac{m}{2}$$

Let us for example m=4, then $j=-1$ and step-size=2. In the first step we transform $f(k)$ to get $c_{-1}(k)$, $k=0.1.....\frac{l}{2}-1$. In the second step we feedback $c_{-1}(k)$ (that is $f(k):=c_{-1}(k)$, $k=0.1.....\frac{l}{2}-1$) on the first $\frac{l}{2}$ input lines and then repeat the first step. The complete iterative procedure can equivalently be represented by the following scheme;

$$\left\{f(k)_{k=0}^{l-1}\right\} \xrightarrow{\text{WT}} \left\{c_{-1}(k)|_{k=0}^{l/2^1-1}\right\} \xrightarrow{\text{WT}} \left\{c_{-2}(k)|_{k=0}^{l/2^2-1}\right\} \ldots\ldots \left\{c_{-r}(k)|_{k=0}^{l/2^r-1}\right\}$$

A switch can be programmed in the feedback ANN to put off the feedback lines and put on the output lines when a required level is achieved. The desired ANN for a signal of length equals 16 is shown in Figure(8.8.1).

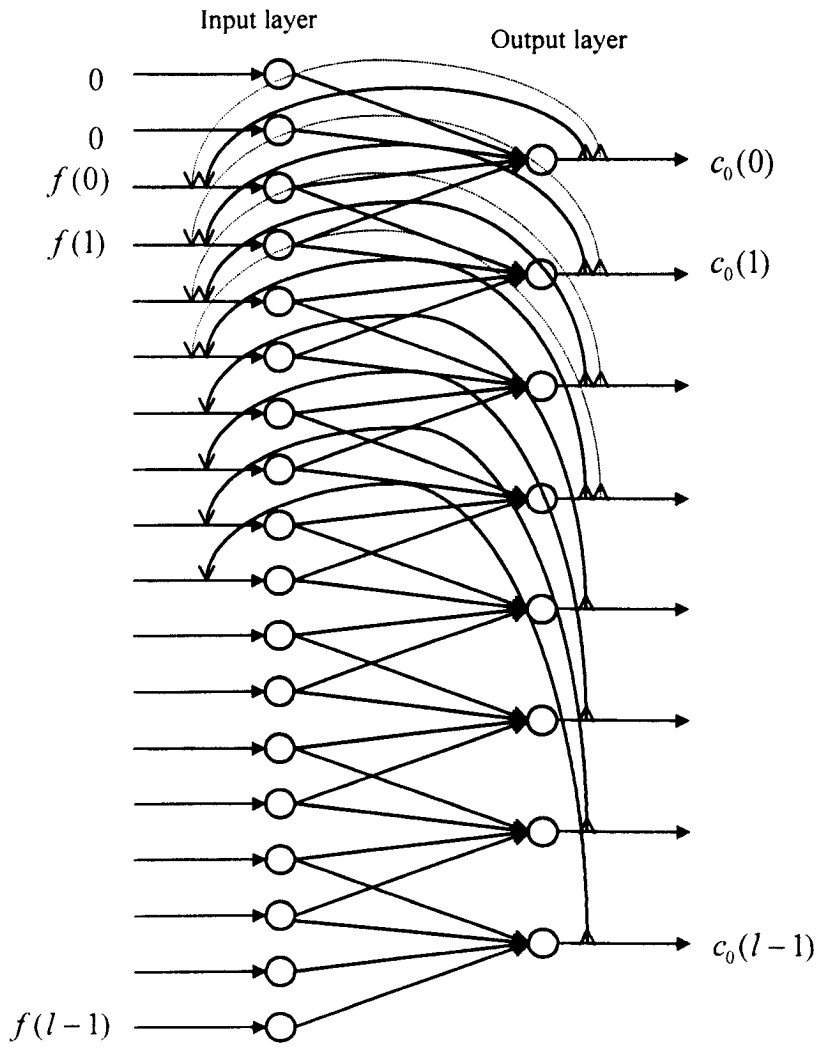


Figure 8.8.1: A wavelet data compressing feedback

A Special One-dimensional and Two-dimensional Wavelet Transform for Noise Reduction and Data Compression

Most of the digital and analogue data generating equipments produce noisy data. The influence of electric and magnetic fields can make data noisy. Similarly, the thundering and lightning of clouds, the mechanical movements of vehicles and industrial machines effects the data during communication. Moreover, the interference of different remote signals always causes noise. So noise reduction is necessary. A latest and efficient technique is the Wavelet Transform.

In order to get a number of targets and due to a wide variety of wavelets, we need to discuss the issue in details. It should be noted that the facts to be presented in this chapter have practically been tested by developing software systems for this purpose. Both, the one-dimensional and two-dimensional software will be described and demonstrated with the help of examples.

9.1 Noise Reduction (Data Smoothing)

Before processing signals and images effectively, we require the data to be sufficiently noise free. That is, it should be sufficiently smoothed. In this section a Wavelet Transform will be developed from the Average Smoothing Filter. A conclusion will be made that " the repeated application of the average smoothing filter is equivalent to a wavelet transform". A software system for the transform is developed and will be presented here. The algorithm and the software is straight forward and leads to develop parallel processing hardware.

9.1.1 A Wavelet Transform deduced from Average Soothing Filter.

An effective and traditional tool of smoothing data is the ***average filter***.

$$\bar{x}_i = \frac{x_i + x_{i+1}}{2} \quad \forall\ i = 1,2,\ldots,n-2\ ;$$

The processing of this filter can be speed up by applying it through a parallel processor. But, what will happen when we need to apply this filter more than once?. Should we install more parallel processors in series?. The answer is No. Because, it is not a wise decision to increase the manufacturing cost and volume of the computer system. The problem is solved interestingly by applying a Wavelet Transform to be deduced from mathematical expression of the repeated application of average smoothing filter.

9.1.2 Origin of Data Smoothing Wavelet Transform is Average Smoothing Filter

Recall the definition of the new wavelet given in Equation(7.2.3) and the wavelets introduced in Section(6.16) which are given below:

(i) $\left\{\frac{1}{4},\frac{2}{4},\frac{1}{4}\right\}$

(ii) $\frac{1}{2}\left\{\frac{1}{8},\frac{2}{8},\frac{3}{8},\frac{4}{8},\frac{3}{8},\frac{2}{8},\frac{1}{8}\right\}$

(iii) $\frac{1}{4}\left\{\frac{1}{16},\frac{2}{16},\frac{3}{16},\frac{4}{16},\frac{5}{16},\frac{6}{16},\frac{7}{16},\frac{8}{16},\frac{7}{16},\frac{6}{16},\frac{5}{16},\frac{4}{16},\frac{3}{16},\frac{2}{16},\frac{1}{16}\right\}$

Let us now apply the average filter twice. First time, we have

$$y_i = \frac{x_i + x_{i+1}}{2} \quad \forall\ i = 1,2,...,n-2\ ;$$

Second time, we have

$$z_i = \frac{y_i + y_{i+1}}{2} \quad \forall\ i = 1,2,...,n-2\ ;$$

The combination of the both gives

$$\begin{aligned} z_i &= \frac{\frac{x_i + x_{i+1}}{2} + \frac{x_{i+1} + x_{i+2}}{2}}{2} \\ &= \frac{x_i + 2x_{i+1} + x_{i+2}}{4} \\ &= \frac{x_i}{4} + \frac{2x_{i+1}}{4} + \frac{x_{i+2}}{4} \\ &= (x_i, x_{i+1}, x_{i+2})(\frac{1}{4},\frac{2}{4},\frac{1}{4}), \quad \forall\ i = 1,2,....,n-2\ ; \end{aligned}$$

This is equivalent to the following one dimensional Wavelet Transform where the set $\left\{\frac{1}{4},\frac{2}{4},\frac{1}{4}\right\}$ is represented by $\{\psi(0),\psi(1),\psi(2)\}$.

$$\begin{bmatrix} z_0 \\ z_1 \\ . \\ . \\ . \\ . \\ . \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} \psi(1) & \psi(2) & 0 & 0 & 0 & . & . & . & . & . & . & . & 0 \\ \psi(0) & \psi(1) & \psi(2) & 0 & 0 & . & . & . & . & . & . & . & 0 \\ 0 & \psi(0) & \psi(1) & \psi(2) & 0 & . & . & . & . & .. & . & . & 0 \\ . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & . & . & . & . & . & . & . & 0 & \psi(0) & \psi(1) & \psi(2) \\ 0 & 0 & . & . & . & . & . & . & . & 0 & 0 & \psi(0) & \psi(1) \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ . \\ . \\ . \\ . \\ . \\ x_{n-1} \end{bmatrix}$$

For the third application of the average filter, we get

$$z_i = (x_i, x_{i+1}, x_{i+2}, x_{i+3})(\frac{1}{8},\frac{3}{8},\frac{3}{8},\frac{1}{8}), \quad \forall\ i = 1,2,...,n-3;$$

For the fourth application of the average filter, we get

$$z_i = (x_i, x_{i+1}, x_{i+2}, x_{i+3}, x_{i+4})[\frac{1}{2}(\frac{1}{8},\frac{4}{8},\frac{6}{8},\frac{4}{8},\frac{1}{8})], \quad \forall\ i = 1,2,...,n-3;$$

Its modified version is given by

$$z_i = (x_i, x_{i+1}, x_{i+2}, x_{i+3}, x_{i+4})[(\frac{1}{9},\frac{2}{9},\frac{3}{9},\frac{2}{9},\frac{1}{9})], \quad \forall\ i = 1,2,...,n-3;$$

From this discussion we can conclude that there is a great resemblance between the repeated application of the average smoothing filter and the wavelets introduced in Section(6.16) and Section(7.2).

Selection of a wavelet depends upon the situation and requirements. A different amount of noise is faced in different situations. For example, different data generating equipments produce different amount of noise ratio. Similarly, different applications, require reduced data.

It is also evident from the theory that the larger the width (number of elements) of the wavelet the more is the noise reduced. In the following section, example programs for the data smoothing wavelet transform are given.

9.2 A One-dimensional Noise Reducing Wavelet Transform and its ANN

It is explained in Chapter5 that an image can be consider as an $n \times n$ matrix of grey levels and then the individual rows can be processed. Here we consider the rows and develop a basic structure of a parallel processor that can smooth the row data. The processor can be termed *as **Noise Reducing Wavelet Transform Artificial Neural Network.***

We want to compute

$$\bar{x}_0 = \frac{2x_0 + x_1}{4}, \quad \bar{x}_{n-1} = \frac{x_{n-2} + 2x_{n-1}}{4}, \text{ and } \bar{x}_i = \frac{x_i + 2x_{i+1} + x_{i+2}}{4} \quad \forall\ i = 1,2,\ldots,n-2,$$

through the ANN given in Figure(9.2.1).

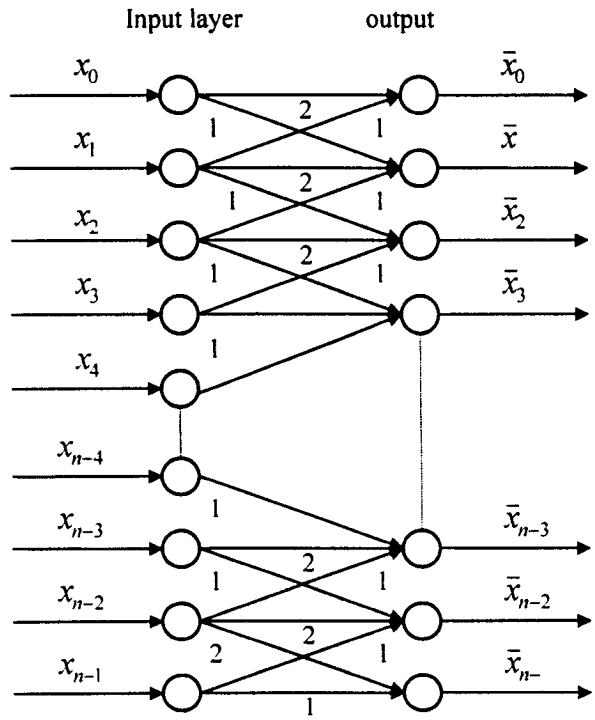


Figure 9.2.1: A Noise Reducing Wavelet Transform ANN.

Depending upon the quality of the digitiser and the nature of the data, one can develop, in the same way a basic structure of the ANN for wider wavelet given as in (ii) and (iii).

Software 9.2: Two programs named **smoth1d1.cpp** and **smoth1d2.cpp** to smooth image data horizontally by applying the wavelets $\frac{1}{4}\{1,2,1\}$ and $\frac{1}{2}\left\{\frac{1}{8},\frac{2}{8},\frac{3}{8},\frac{4}{8},\frac{3}{8},\frac{2}{8},\frac{1}{8}\right\}$ respectively are presented in Appendix(4.1) and Appendix(4.2). Their performance can be seen in Table(9.2(i)-(iii)).

Table(9.2(i)) First two rows of a black and white image mona.dat

102 97 96 101 100 103 103 100 107 107 100 104 102 98 97 103
105 95 98 92 88 99 102 102 99 101 101 97 92 98 95 91
97 95 99 98 95 97 99 98 105 108 106 102 100 105 89 107
89 98 101 103 101 107 114 117 115 123 120 122 131 130 122 129
132 138 138 137 141 130 124 120 126 127 127 118 114 115 114 113
111 116 114 118 116 118 119 118 115 128 130 136 129 127 127 117
119 116 119 121 116 128 120 117 116 118 116 113 124 117 121 123
118 120 117 118 120 121 122 119 119 113 118 122 123 118 118 118
96 104 107 108 102 103 103 97 105 113 103 111 111 107 104 103
101 97 104 109 108 104 100 105 99 101 96 104 105 105 103 100
100 104 106 110 103 100 102 105 110 118 111 107 102 105 96 102
99 103 109 114 111 113 116 117 113 115 122 127 131 127 122 125
128 131 133 138 139 135 131 132 133 130 133 122 127 117 121 123
120 121 121 115 117 121 121 128 120 133 132 135 132 130 129 126
129 121 123 132 117 119 120 124 117 122 117 118 127 119 118 118
118 120 120 117 122 125 117 123 119 115 126 127 126 128 124 125

Table(9.2(ii)) The data after execution of smoth1d1.cpp

100 98 97 99 101 102 102 102 105 105 102 102 101 98 98 102
102 98 95 92 91 97 101 101 100 100 100 96 94 95 94 93
95 96 97 97 96 97 98 100 104 106 105 102 101 99 97 98
95 96 100 102 103 107 113 115 117 120 121 123 128 128 125 128
132 136 137 138 137 131 124 122 124 126 124 119 115 114 114 112
112 114 115 116 117 117 118 117 119 125 131 132 130 127 124 120
117 117 118 119 120 123 121 117 116 117 115 116 119 119 120 121
119 118 118 118 119 121 121 119 117 115 117 121 121 119 118 118
98 102 106 106 103 102 101 100 105 108 107 109 110 107 104 102
100 99 103 107 107 104 102 102 101 99 99 102 104 104 102 100
101 103 106 107 104 101 102 105 110 114 111 106 104 102 99 99
100 103 108 112 112 113 115 115 114 116 121 126 129 126 124 125
128 130 133 137 137 135 132 132 132 131 129 126 123 120 120 121
121 120 119 117 117 120 122 124 125 129 133 133 132 130 128 127
126 123 124 126 121 118 120 121 120 119 118 120 122 120 118 118
118 119 119 119 121 122 120 120 119 118 123 126 126 126 125 124

Table(9.2(iii)) The data after execution of smoth1d2.cpp

99 98 99 99 100 101 102 103 103 103 103 102 101 100 100 100
99 97 96 95 95 96 98 100 100 99 98 97 96 95 94 94
95 96 96 97 97 97 99 100 102 104 104 102 101 100 98 98
97 98 100 101 104 108 111 114 117 120 122 124 126 127 128 130
132 134 136 136 134 130 127 125 124 123 122 119 117 115 114 113
113 114 115 116 117 117 118 119 121 125 127 129 129 126 124 121
119 118 118 119 120 120 119 118 117 117 116 117 118 119 120 120
119 119 118 119 119 120 119 119 118 118 118 119 119 119 119 118
101 103 104 104 103 103 102 103 104 106 107 108 108 106 104 103
102 102 103 104 105 104 103 101 100 100 101 102 102 103 102 102
102 103 104 104 104 103 104 106 108 110 109 107 104 102 101 101
102 104 107 110 112 113 114 115 116 118 121 124 125 126 126 126
128 130 133 134 135 134 133 132 131 130 128 126 124 122 121 121
120 120 119 118 119 120 122 124 126 129 130 131 131 130 128 127
126 125 124 123 122 121 120 120 119 120 119 120 120 119 119 119
118 119 119 119 120 121 120 120 120 121 123 124 125 126 125 125

9.3 Two-dimensional Data Smoothing Wavelet Transform and its ANN

A black and white image consists of two-dimensional data. Corresponding to one dimensional wavelet $\frac{1}{4}\{1,2,1\}$, we have the following two-dimensional wavelet

$$\frac{1}{10}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 2 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \qquad (9.3.1)$$

Similarly, corresponding to the one dimensional wavelet $\frac{1}{9}\{1,2,3,2,1\}$, we have the following two dimensional wavelet

$$\frac{1}{35}\begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 2 & 2 & 2 & 1 \\ \hline 1 & 2 & 3 & 2 & 1 \\ \hline 1 & 2 & 2 & 2 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline \end{array} \qquad (9.3.2)$$

One can have several two-dimensional wavelet in the same way. Again, the larger the width of the wavelet, the more is the smoothing of data.

However, for the wavelet given in Relation(9.3.1) we proceed for two dimensional wavelet transform as follows;

If x_{ij} represents an element of the image then the smoothed element, say, y_{ij} is given by

$$y_{i,0} = \{(x_{i-1,0} + x_{i-1,1}) + (2 * x_{i,0} + x_{i,1}) + (x_{i+1,0} + x_{i+1,1})\} / 7 ,$$

$$y_{ij} = \{(x_{i-1,j-1} + x_{i-1,j} + x_{i-1,j+1}) + (x_{i,j-1} + 2 * x_{i,j} + x_{i,j+1}) + (x_{i+1,j-1} + x_{i+1,j} + x_{i+1,j+1})\} / 10 ,$$

$$y_{i,n-1} = \{(x_{i-1,n-2} + x_{i-1,n-1}) + (x_{i,n-2} + 2 * x_{i,n-1}) + (x_{i+1,n-2} + x_{i+1,n-1})\} / 7 , \quad i = 1,2,\ldots,n-2 \text{ and } j = 0,1,2,\ldots,n-1.$$

The **parallel processing** can be shown as follows;

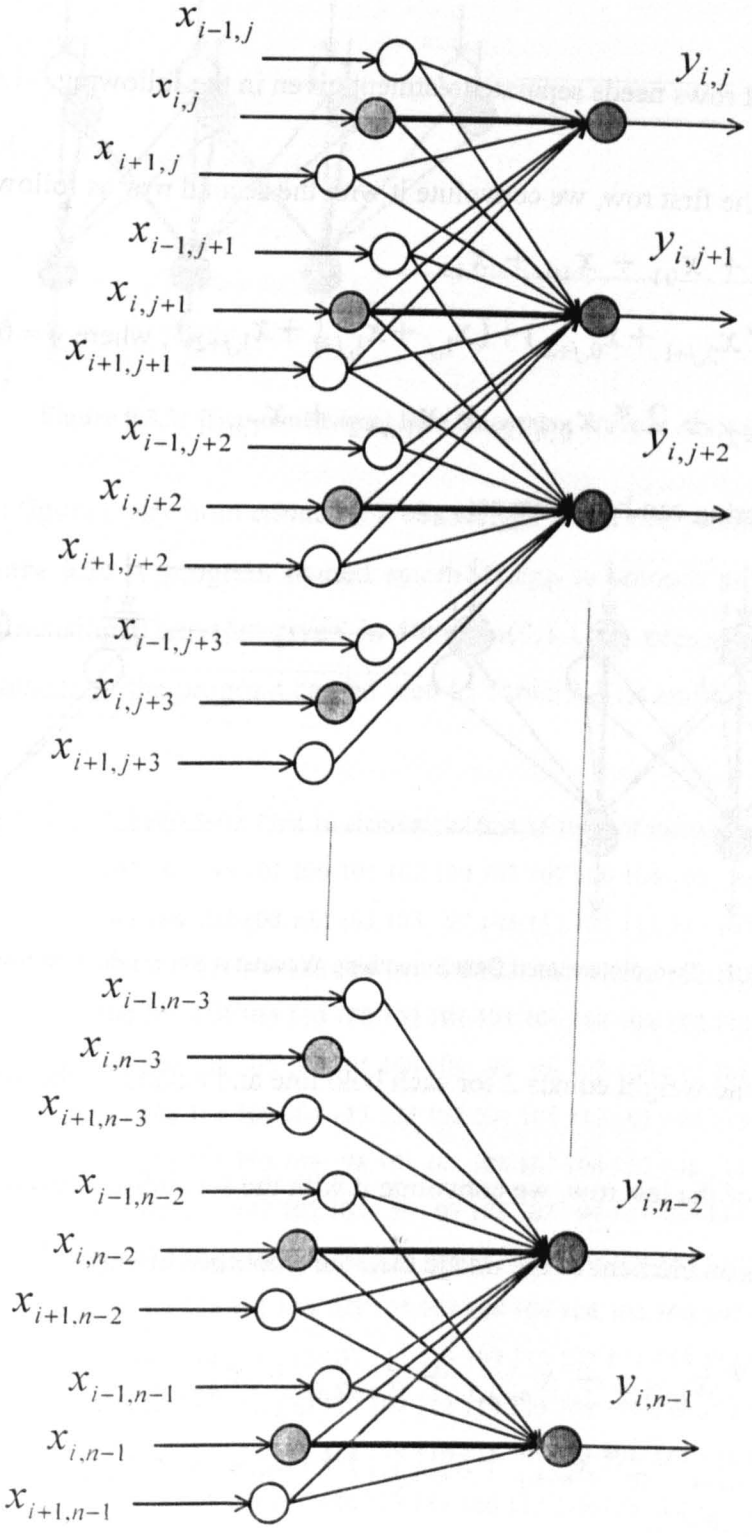


Figure 9.3.1: Two-dimensional Data Smoothing Wavelet ANN.

In Figure(9.3.1) every bold connection has weight 2 and every other connection have weight 1.

The first and the last rows needs separate treatment given in the following.

First row:

To smooth data of the first row, we convolute it with the second row as follows;

$$y_{00} = 2 * x_{00} + x_{01} + x_{10} + x_{11}$$

$$y_{0j} = (x_{0,j} + 2 * x_{0,j+1} + x_{0,j+2}) + (x_{1,j} + x_{1,j+1} + x_{1,j+2}), \text{ where } j = 0,1,2,\ldots,n-3$$

$$y_{0,n-1} = x_{0,n-2} + 2 * x_{0,n-2} + x_{1,n-2} + x_{1,n-1}$$

The **parallel processing** can be shown as;

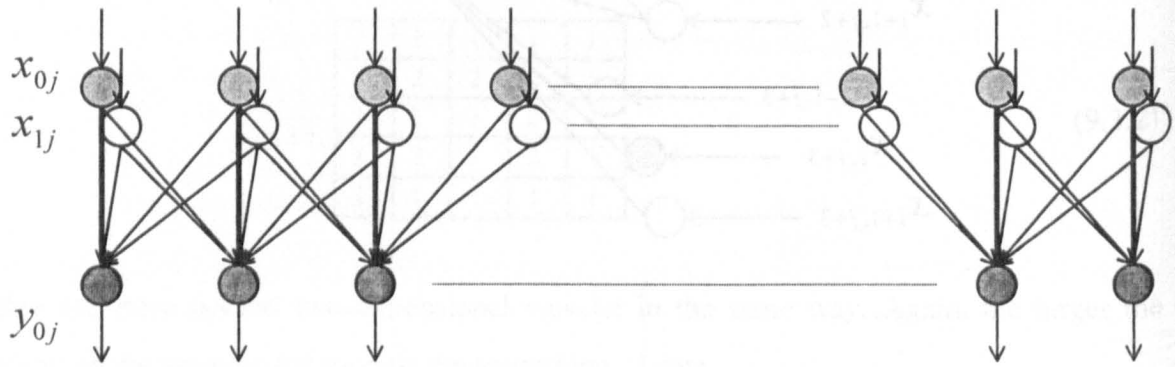


Figure 9.3.2: Two-dimensional Data Smoothing Wavelet ANN for the first row.

In Figure(9.3.2) the weight equals 2 for each bold line and equals 1 otherwise.

Last row:

To smooth data of the last row, we convolute it with the second-last row as follows;

If x_{ij} represents an element of the image then the smoothed element y_{ij}, say, is given by

$$y_{n-1,0} = 2 * x_{n-1,0} + x_{n-1,1} + x_{n-2,0} + x_{n-2,1}$$

$$y_{n-1,j} = (x_{n-1,j-1} + 2 * x_{n-1,j} + x_{n-1,j+1}) + (x_{n-2,j-1} + x_{n-2,j} + x_{n-2,j+1}),$$

where $j = 1,2,\ldots,n-2$.

$$y_{n-1,n-1} = x_{n-1,n-2} + 2 * x_{n-1,n-2} + x_{n-2,n-2} + x_{n-2,n-1}$$

The **parallel processing** can be shown as given below:

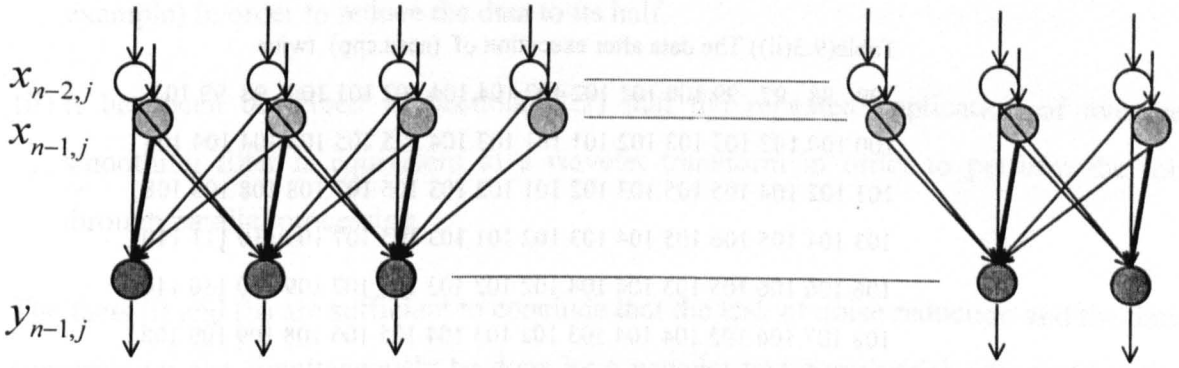


Figure 9.3.3: Two-dimensional Data Smoothing Wavelet ANN for the last row.

In this figure every bold connection has weight 2 and every other connection have weight

Software 9.3: A program named **smoth2d1.cpp** to smooth an image data by using the two-dimensional wavelet given in Relation(9.3.1) is presented in Appendix(4.3). The performance of the program can be seen in Table(9.3(i)) and Table(9.3(ii)).

Table(9.3(i)) First 16 elements of first 16 rows of mona1.dat

102 97 96 101 100 103 103 100 107 107 100 104 102 98 97 103
96 104 107 108 102 103 103 97 105 113 103 111 111 107 104 103
99 100 111 103 108 106 101 98 103 100 107 110 112 107 113 113
104 103 110 109 110 108 102 101 101 104 108 106 112 113 115 113
107 104 108 105 100 105 108 109 99 99 102 106 112 108 110 116
115 106 109 104 104 113 103 102 104 101 113 103 109 113 114 106
114 111 105 105 109 98 101 101 108 107 106 112 108 111 107 106
113 106 103 107 107 104 99 97 108 107 99 101 109 111 106 105
113 112 106 110 106 98 107 104 111 104 110 107 105 112 115 112
115 104 104 111 118 105 104 104 108 109 104 103 106 107 108 112
110 114 114 116 115 107 111 110 109 110 107 111 113 114 118 111
117 116 116 118 120 112 109 112 119 115 108 115 116 120 111 109
121 119 123 125 118 114 118 116 115 109 121 124 117 122 124 120
124 124 130 120 123 120 117 117 120 117 119 126 122 122 114 118
132 132 131 125 124 125 117 119 122 127 130 129 117 129 130 129
136 134 133 130 128 117 124 126 116 122 128 127 127 128 127 129

Table(9.3(ii)) The data after execution of (aaaa.cpp) twice

99 98 97 99 100 101 102 102 104 104 102 101 100 98 99 101
100 100 102 103 103 102 101 101 103 104 105 105 105 104 104 104
101 102 104 105 105 103 102 101 102 103 105 107 108 108 108 108
103 104 105 106 105 104 103 102 101 103 105 107 109 110 111 110
106 106 106 105 105 104 104 102 102 103 105 107 109 110 110 110
108 107 106 105 104 104 103 102 103 104 105 106 108 109 109 108
109 108 106 105 104 103 102 102 104 105 105 106 108 108 108 107
109 108 106 106 105 103 102 103 104 105 105 105 107 108 108 107
109 108 107 107 106 104 103 104 105 106 105 105 107 108 109 109
110 109 109 110 109 107 106 106 107 107 106 107 108 109 110 111
112 112 113 113 112 110 109 109 109 109 109 109 111 112 112 113
116 116 116 116 115 113 112 112 112 112 112 114 115 115 115 115
120 120 120 120 118 116 115 115 115 115 117 118 119 119 118 118
125 125 124 123 121 119 117 117 117 119 120 122 122 121 121 121
129 129 128 125 122 121 119 119 120 122 123 124 124 124 124 125
132 131 130 127 124 122 121 122 123 124 124 125 125 127 128 129

9.4 Data Compressing Wavelet Transform and it Origin

It has become possible to transmit video data like the audio data. Video data, being huge compared with the audio data, needs reduction to produce a matching with audible speech.

Some techniques are used to reduce the data before transmitting it. The reduction is made in such a way that the technique used will ensure the reconstruction of the missing data at the receiving end. A latest and efficient technique is the Wavelet Transform.

(i) A basic technique of downsizing the data is "down sampling". According to which data is reduced to half, (for example) by discarding each alternative element. This idea is required to be inherited in wavelet transform. That is, the wavelet (wavelet

window) should be moved along the signal of data with step size equal to 2 (for example) in order to reduce the data to its half.

(ii) It has been described in Section(9.1.2) that the repeated application of average smoothing filter is equivalent to a wavelet transform in order to perform the job through parallel processing.

The facts (i) and (ii) are sufficient to conclude that the task of noise reduction and the data compression can simultaneously be done by a wavelet transform which can perform the job of the following series of processes;

$$M \text{ * average filter applied once + } N \text{ * down sampling by 2}$$

where M is the number of repetitions required to smooth the data sufficiently,

and N is the number of repetitions required to reduce the data by a factor of 2^N.

9.5 One-dimensional Data Compressing Wavelet Transform and its Software

Consider the wavelet $\frac{1}{10}\{1,4,4,1\}$ then the first round of wavelet transform, described in Chapter7, of a signal $f(x)$ is given by

$$\begin{bmatrix} c_{10} \\ c_{11} \\ \cdot \\ \cdot \\ \cdot \\ c_{1,n/2^1-1} \end{bmatrix} = \begin{bmatrix} .4 & .1 & 0 & 0 & 0 & \cdots & & & & & & & 0 \\ .1 & .4 & .4 & .1 & 0 & 0 & 0 & \cdots & & & & & 0 \\ 0 & 0 & .1 & .4 & .4 & .1 & 0 & \cdots & & & & & 0 \\ \cdots & & & & & & & & & & & & \\ \cdots & & & & & & & & & & & & \\ 0 & \cdots & & & & & & & 0 & .1 & .4 & .4 & .1 \\ 0 & \cdots & & & & & & & 0 & 0 & 0 & .1 & .4 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \cdot \\ \cdot \\ \cdot \\ f_{n-1} \end{bmatrix}$$

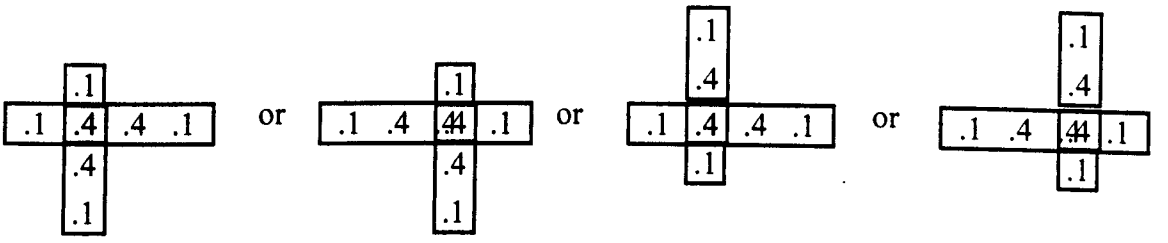
And the first round of inverse wavelet transform is given by

$$\begin{bmatrix} f_0 \\ f_1 \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ f_{n-1} \end{bmatrix} = \begin{bmatrix} .4 & .1 & 0 & . & . & . & 0 & 0 \\ .1 & .4 & 0 & . & . & . & . & . \\ 0 & .4 & .1 & . & . & . & . & . \\ 0 & .1 & .4 & . & . & . & . & . \\ 0 & 0 & .4 & . & . & . & . & . \\ 0 & 0 & .1 & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & 0 & 0 \\ 0 & . & . & . & . & . & .1 & 0 \\ 0 & . & . & . & . & . & .4 & 0 \\ 0 & . & . & . & . & . & .4 & .1 \\ 0 & . & . & . & . & . & .1 & .4 \end{bmatrix} \begin{bmatrix} c_{10} \\ c_{11} \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ . \\ c_{1,n/2^1-1} \end{bmatrix}$$

Software 9.5: To demonstrate the performance of the wavelet $\frac{1}{10}\{1,4,4,1\}$ for data compression plus noise reduction, two programs named **twave22.cpp** and **twave222.cpp** has been presented in Appendix(4.4) and Appendix(4.5) respectively.

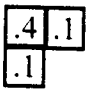
9.6 A two-dimensional Data Compressing Wavelet Transform and its Sotware

The wavelet transform in the above section has been computed by applying the wavelet { .1 .4 .4 .1 } once vertically and then horizontally. From hardware point of view, this strategy leads to the development of an Artificial Neural Network with two hidden layers. Instead we are interested in a single hidden layer ANN. The two layers can be unified and result can be improved by applying a wavelet window of the form

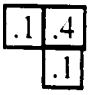


In applying a window for a wavelet transform, the zero padding was found to be the best. Instead to pad zeros in each data file to be processed, it should be preferred (for details see Section(8.2) to reduce the wavelets as given below:

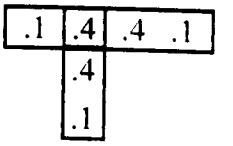
(i) for the left-upper corner of the image, nonzero part of the window is



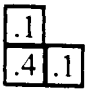
(ii) for the right-upper corner of the image, we apply the window



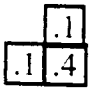
(iii) for the first row excluding its corners, we apply the window



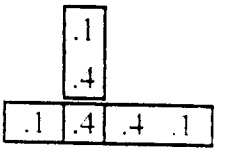
(iv) for the left-bottom corner of the image, we apply the window



(v) for the right-bottom corner of the image, we apply the window



(vi) for the last row excluding its corners, we apply the window



However, it is more advantageous if we apply the wavelet given in Figure(9.6.1) and Figure(9.6.2).

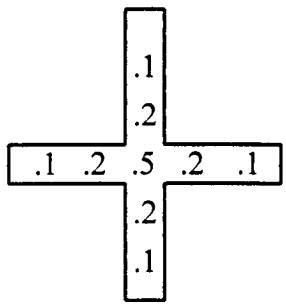


Figure a)

.5	.1	.1	.5
.1	.5	.5	.1
.1	.5	.5	.1
.5	.1	.1	.5

Figure b)

Figure 9.6.1: a) A Two-dimensional Wavelet for Data Compression and b) A Two-dimensional Wavelet for Data Expansion.

.1	.1	.1	.1
.1	.4	.4	.1
.1	.4	.4	.1
.1	.1	.1	.1

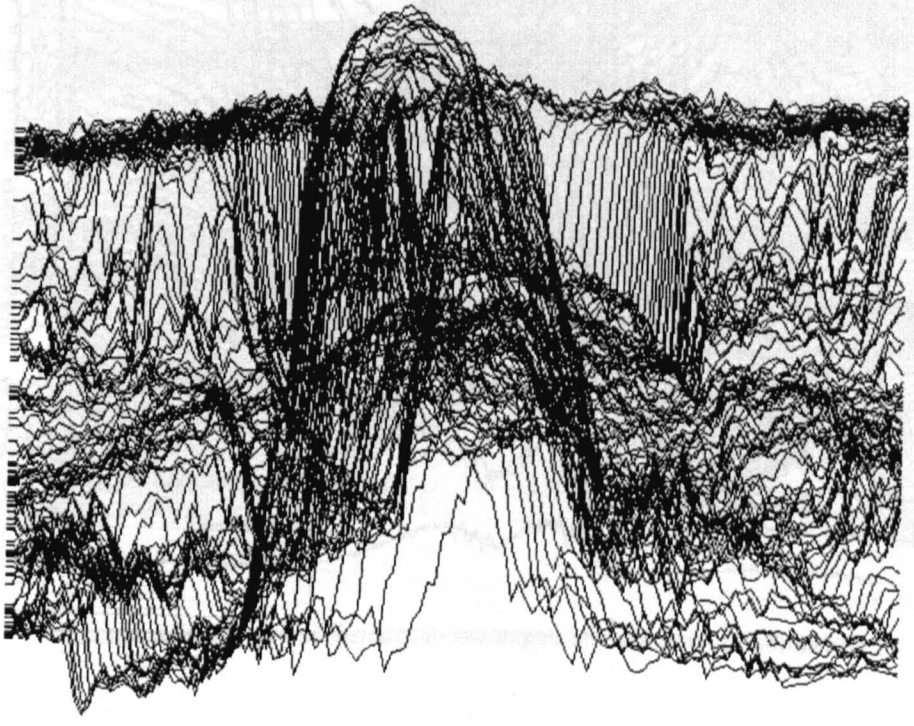
Figure a)

.4	.1	.1	.4
.1	.4	.4	.1
.1	.4	.4	.1
.4	.1	.1	.4

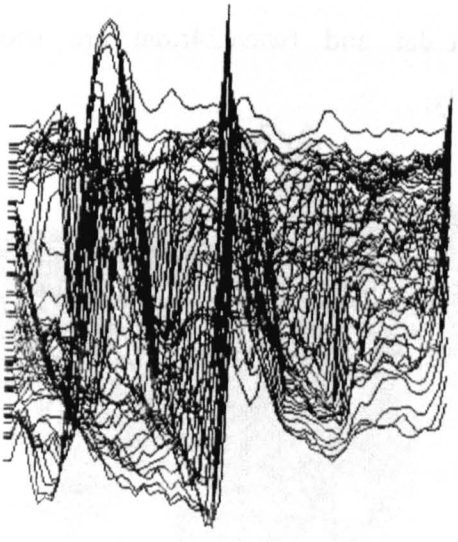
Figure b)

Figure 9.6.2: a) A Two-dimensional Wavelet for Data Compression and b) A Two-dimensional Wavelet for Data Expansion.

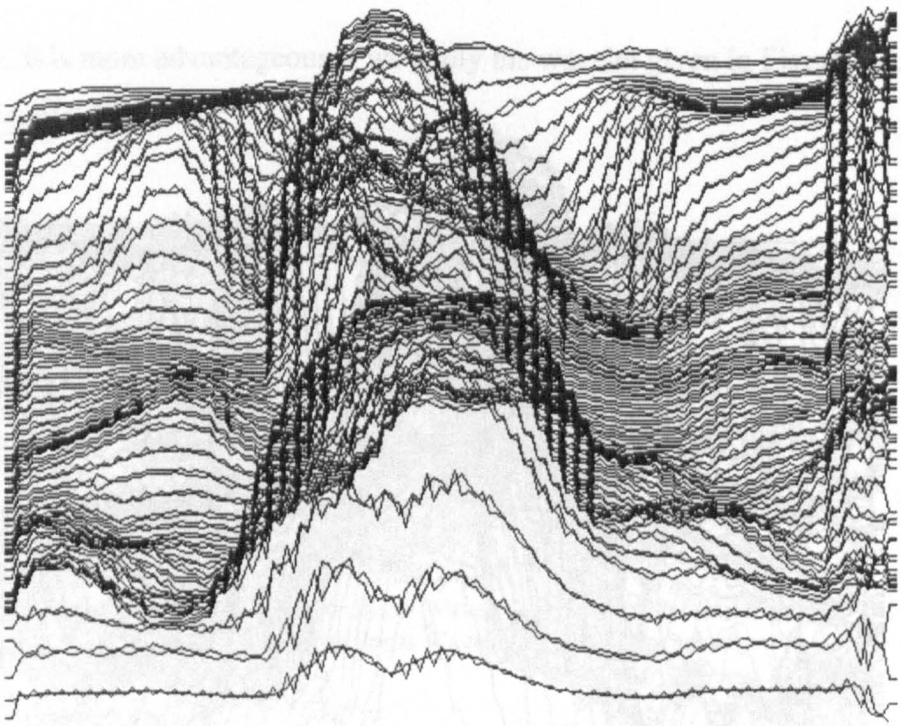
Software 9.6.1: A coded program named **twave23.cpp** has been presented in Appendix(4.6) to demonstrate the performance of the two-dimensional wavelet given in Figure(9.6.1a) and its inverse wavelet given in Figure(9.6.1b) is applied to compute the inverse wavelet transform for data compression plus noise reduction. The graph of the input image mona1.dat is shown in Figure(9.6.3a) and the graphs of the output (compressed) image twave23c.dat and the output (reconstructed) image twave23r.dat are shown in Figure(9.6.3b) and Figure(9.6.3c), respectively.



Figure(9.6.3a): Graph of the input image mona1.dat

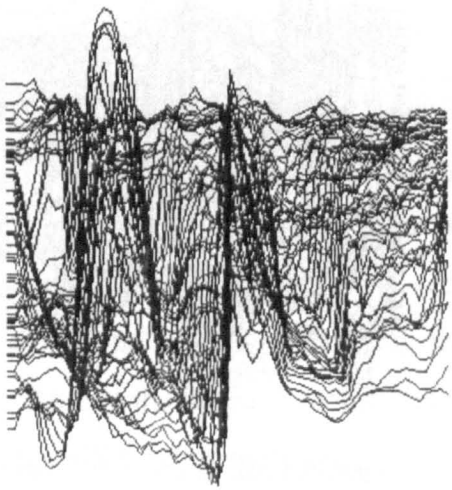


Figure(9.6.3b): Graph of the output (compressed) image twave23c.dat

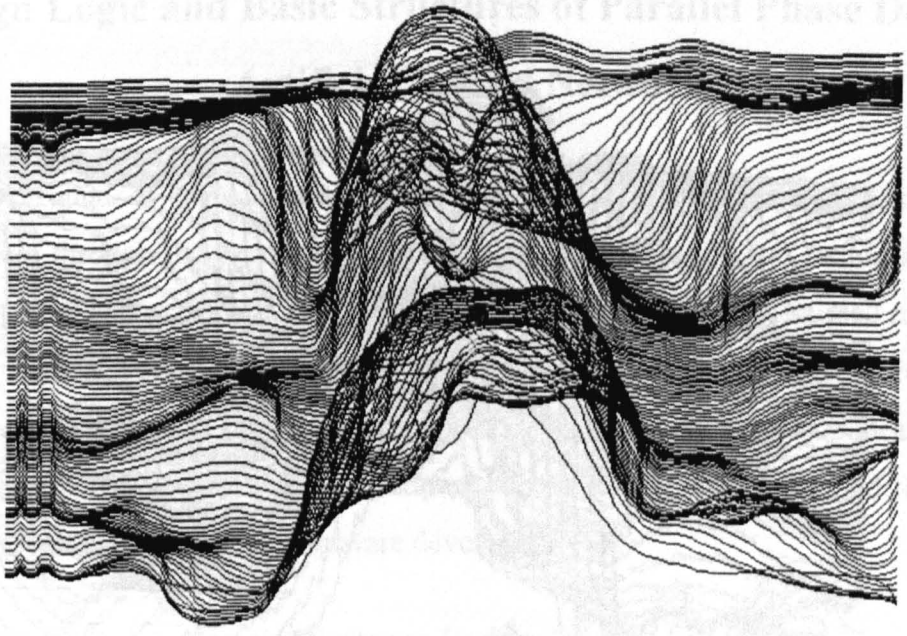


Figure(9.6.3c): Graph of the output (reconstructed) image twave23r.dat

Software 9.6.2: A coded program named **twave24.cpp** has been presented in Appendix(4.7) to demonstrate the performance of the two-dimensional wavelet given in Figure(9.6.2a) and the inverse wavelet transform is computed by applying the wavelet $\{ .1 \quad .4 \quad .4 \quad .1 \}$ once horizontally and then vertically to image data. The graphs of the output images twave24c.dat and twave24r.dat are shown in Figure(9.6.4a) and Figure(9.6.4b), respectively.

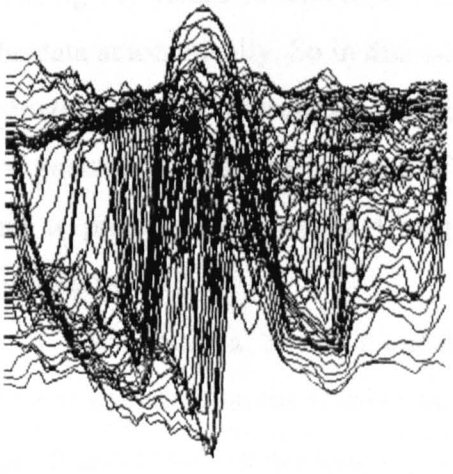


Figure(9.6.4a): Graph of the output (compressed) image twave24c.dat

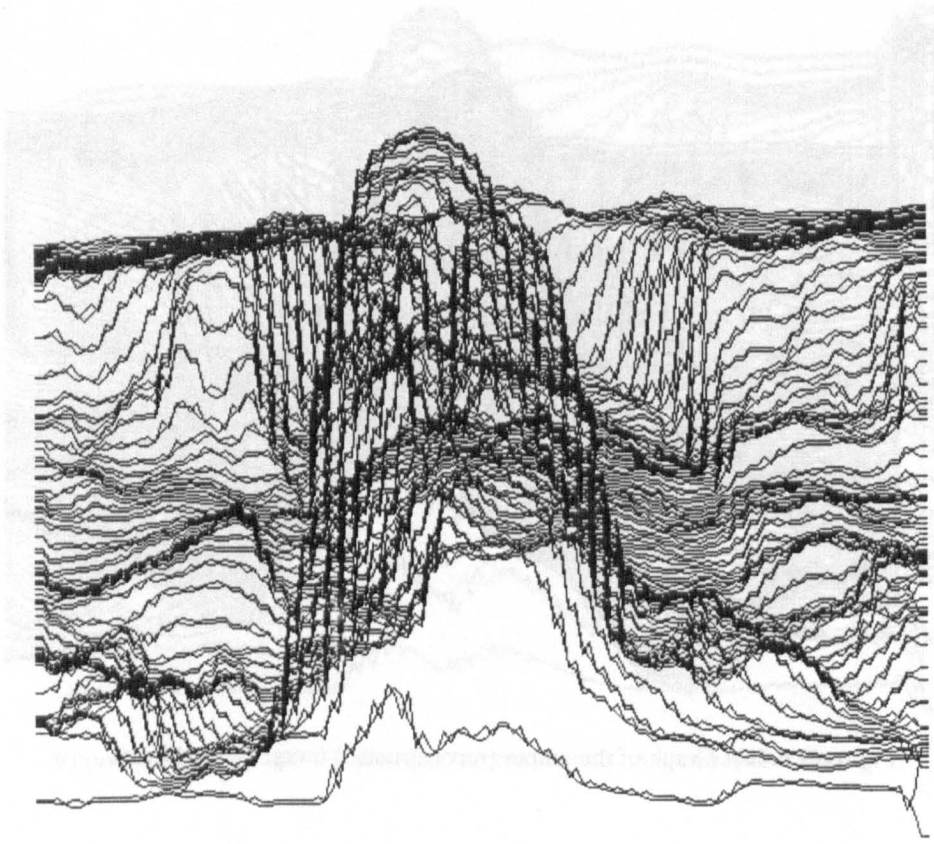


Figure(9.6.4b): Graph of the output (reconstructed) image twave24r.dat

Software 9.6.3: A coded program named **twave25.cpp** has been presented in Appendix(4.8) to demonstrate the performance of the two-dimensional wavelet given in Figure(9.6.2a) and its inverse wavelet given in Figure(9.6.2b). The graphs of the output images twave25c.dat and twave25r.dat are shown in Figure(9.6.5a) and Figure(9.6.5b), respectively.



Figure(9.6.5a): Graph of the output (compressed) image twave25c.dat



Figure(9.6.5b): Graph of output (reconstructed) image twave25r.dat

Design Logic and Basic Structures of Parallel Phase Detecting Artificial Neural Networks

The background of this chapter is presented in Chapter5 where the task of pattern recognition through the prominent features like peaks of cusps and bottoms of troughs was suggested and explained. In this chapter this concept will be discussed more closely such that for each stage of computation a parallel hardware can be developed and used. This chapter is divided into three groups of sections: the first group deals with the concept and software developing for sequential computers, the second and the third are devoted to parallel processing hardware developing.

10.1 Concept Developing and Software Developing

In this section, the complete procedure will be presented from sequential point of view. The effectiveness and accuracy of the procedure will be tested by presenting coded examples. It should be noted that the software will be executable on sequential processing computers while the hardware to be developed to perform the task of this software will have parallel processing. Stepwise key points are:

(i) Smooth the image data by using a suitable two-dimensional Wavelet Transform.

(ii) If required, compress(or expand) the image data by using a suitable two-dimensional Data Compressing(or Expanding). It should be remember that a data compressing wavelet transform also smoothes the data automatically. So in this case step (i) is not required.

(iii) Apply peaks discriminating (phase detecting) algorithm to find the positions, amplitudes, and number of peaks of cusps and bottoms of troughs of each row of the image data. This algorithm is preliminary given in Chapter5. Developing its parallel hardware will be started from Second(10.5).

(iv) Data can be reduced by developing a binary number to represent, the ordering structure of occurrence of cusps and troughs, the number of waves, the widths of waves, etc. of each row of an image. This step will be discussed in Section(10.4) of this Chapter.

The successful completion of all of the above steps makes it possible to classify a huge image accurately by using only a few integers extracted from it.

Software 10.1.1: (Smoothing of the image Data)

The data file "mona.dat" is a binary file of 128x128 black and white image of Monalisa having 256 grey levels ranging (0-255) and was digitised at McCleod Institute of Simulation Sciences, DeMontfort University Leicester, England.

As it has already been explained in Chapter8 that for a particular application, we smooth an image data up to the suitable extent. This fact prevents us to declare a Data Smoothing Wavelet Transform to be universal. In this example, the amount of noise from the image "mona.dat" requires five times repeated execution of the main block of program "smooth11.exe". Although an equivalent wavelet transform can be developed to avoid these repetitions, yet we use this program as it is. Because this is not the issue of this chapter. But it should be noted that this would have to be done to perform the task through a Single-Hidden-Layer Feed-Forward Data-Smoothing ANN. See the software program named **smooth11.cpp** in Appendix(5.1).

Software 10.1.2: (Detecting peaks and setting other data equals zero)

The program named **tfpsdz.cpp**(Test the Finding of Peaks of a Signal and setting other Data elements equals Zero) is developed to detect the peaks of cusps and bottoms of troughs of a signal and to set all other data elements equals 0. The program is listed in Appendix(5.2) and its outputs are shown in Figure(10.1).

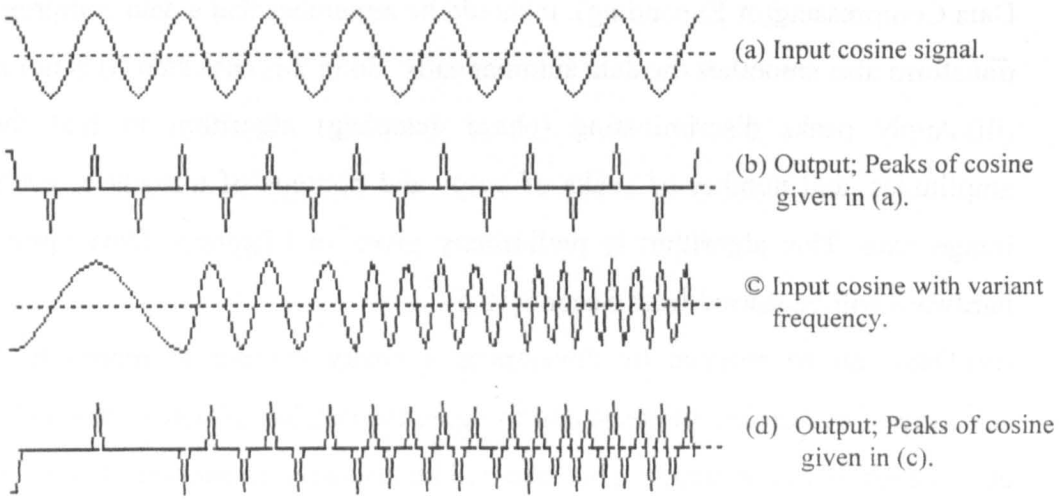


Figure 10.1: Outputs of tfpsdz.cpp to demonstrate detecting peaks of the input signal and setting its all other elements equals zero.

Software 10.1.3: (Reconstruction of Data Between Peaks of Signals with Software Check)

The program named **tfpsdzr.cpp** is developed for signals to reconstruct their data elements that were set equals zero by the program tfpsdz.cpp given in Software(10.1.2). Reconstruction is done by using a flexible wavelet transform, basically introduced in Section(8.6), which automatically and instantly decides the width and height of the wavelet. It is listed in Appendix(5.3) and its outputs are shown in Figure(10.2).

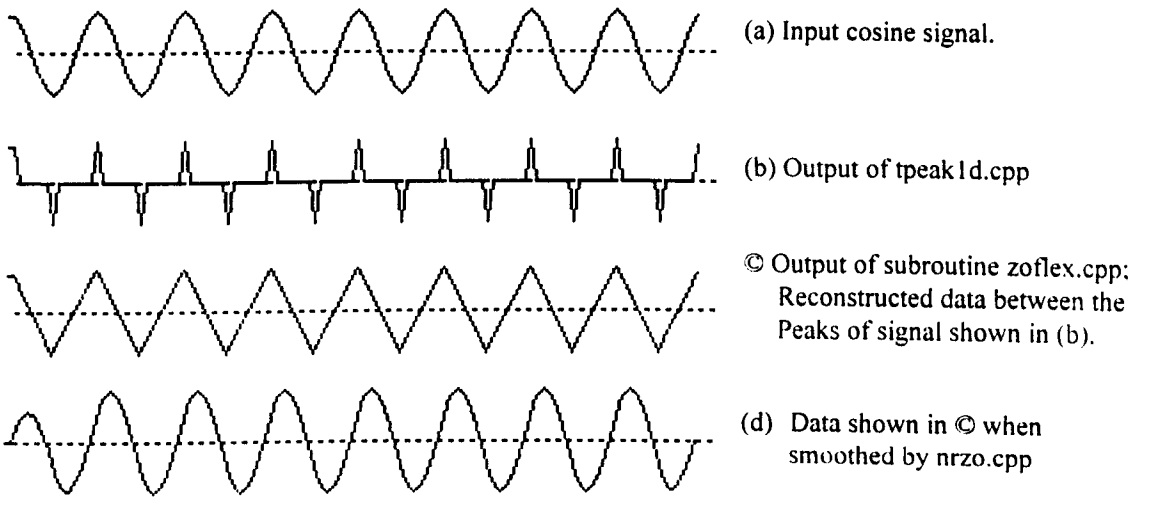


Figure 10.2: Outputs of tfpsdzr.cpp to demonstrate the construction of data elements between peaks that were set equals zero by the program tpeak1d.cpp.

10.2 A Modification of New Zoom-out Transform and Reconstruction of Data

Observe that the sharp corners of the reconstructed signal shown in Figure(10.2©) has required data smoothing subsequently. This minor drawback is due to the triangular shape of the wavelet used in reconstruction. Recall that this wavelet was basically introduced in Section(7.2). Then it resulted in Section(8.6) as a consequence of the modification in CWT made in Section(8.1). Moreover, it resembles with the wavelet resulted in Section(9.1.2) from the repeated application of the smoothing filter. Above of all, being the identity wavelet, it is the simplest possible one. But even then its shape needs to be modified from triangular to Gaussian.

Suppose that either the idea of this modification is not clear yet or otherwise we do not want to use the existing mathematical definition in having a Gaussian wavelet. The two transforms, the reconstruction and the smoothing, are non-commutative and can be unified by unifying the two wavelets used. Mathematically, we can express the problem as:

Let Ψ and Φ are the two non-commutative matrices then find a matrix $\mathbf{G}$ such that

$$\Psi\Phi F = \mathbf{G}F \tag{10.2.1}$$

where F is the column matrix consisting of the elements of the given signal as shown in Figure(10.2(b)); Φ and Ψ are the matrices to be developed from wavelets used in the reconstructing and smoothing transforms, respectively. The matrices Ψ and Φ can not be developed in the usual way used in Chapters(7-9) due to the facts given below:

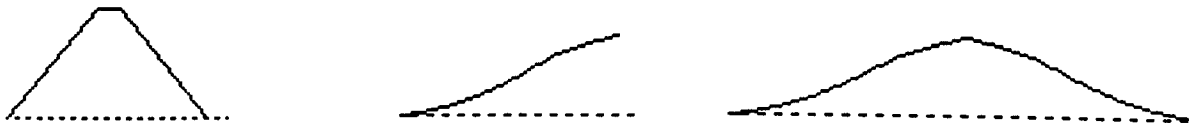
- The matrix Ψ, being the second and non-commuting, can not be developed independently. Instead, its widths of rows and columns will be in accordance with that of the matrix Ψ such that the matrix multiplication be preserved.
- Since the reconstructing transform instantly sets the width of the wavelet equals the positive difference of the indices of a couple (peak, bottom) or (bottom, peak), therefore, the matrix Ψ can not be developed in the usual way.

The roots of the reconstructing transform, and hence of the reconstructing wavelet of flexible width, are in Section(8.6). There a matrix has been developed from a double wavelet. The shape of the matrix could not reflected both, the advantages of its producing double wavelet and unusualness of the way of construction. But here the solution of the problem given in Relation(10.2.1) requires to elaborate it.

It has been assessed that the solution of the problem lies in smoothing of the wavelet (triangular wavelet) to be used in reconstructing transform by applying the data smoothing wavelet transform which uses the triangular wavelet as well. The resulting smoothed wavelet will be a Gaussian like wavelet which should be used to develop the required matrix $\mathbf{G}$. Developing the matrix $\mathbf{G}$ from the double wavelet expressed in Section(8.6) is equivalent to the fact that the matrix be developed either from the first half parts of the Gaussian wavelet or from the logistic functions.

Software 10.2.1: (Testing of a Different Development of Gaussian)

A program **tddgauss.cpp** has been developed and listed in Appendix(5.4). Its outputs are shown in Figure(10.3).



(a) Input Triangular Wavelet. (b) Half Gaussian (Logistic type) (c) Output Gaussian Wavelet.

Figure 10.3: Outputs of tddgauss.cpp

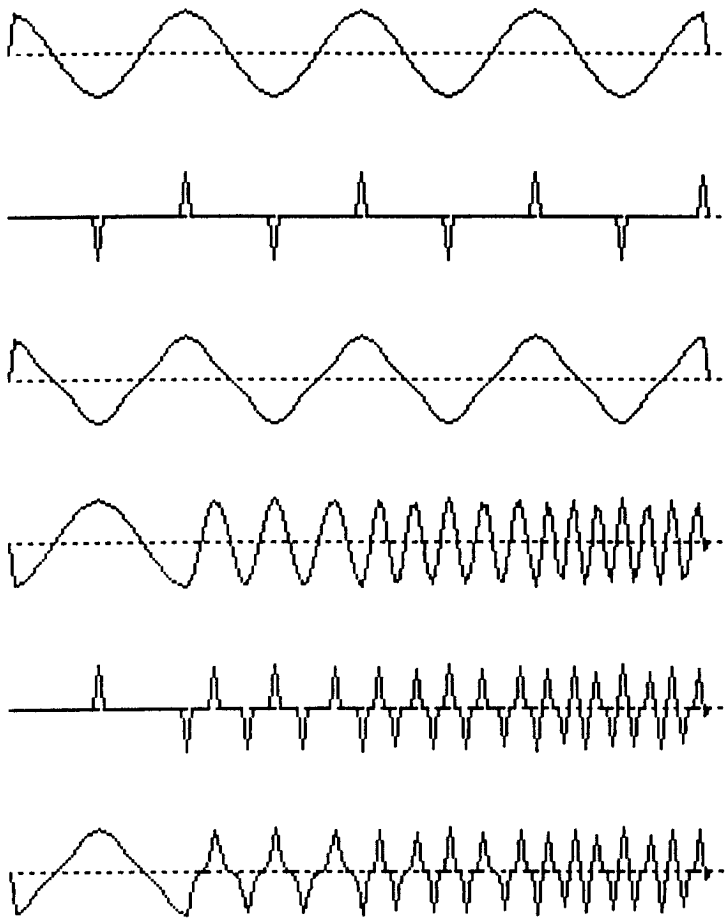
Let m_i $(i = 1,2,...,n)$ denotes the positive difference of the indices of the i^{th} couple (peak, bottom) or (bottom, peak), as were the case. Then the reconstructing plus smoothing matrix **G** can be written as

$$\mathbf{G} = \begin{bmatrix} \boxed{\begin{matrix} m_1 & 0 & 0 & 0 & 0 \\ m_1-1 & 1 & 0 & 0 & 0 \\ m_1-2 & 2 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & m_1-1 & 0 & \cdots & \end{matrix}} & & & \\ & \boxed{\begin{matrix} m_2 & 0 & 0 & \cdots \\ m_2-1 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & m_2-1 & 0 & \cdots \end{matrix}} & & \\ & & \ddots & \\ & & & \boxed{\begin{matrix} m_n & 0 & \cdots \\ m_n-1 & 1 & \cdots \\ \vdots & \vdots & \vdots \\ 1 & m_n-1 & 0 \end{matrix}} \end{bmatrix}$$

Hence, we have $\Psi\Phi = \mathbf{G}$ by using first half of Gaussian to develop a double wavelet for each couple (peak, bottom) or (bottom, peak). Observe that each section of **G** enclosed in a rectangle consists of a double wavelet. One way to the other a column of a section makes a half Gaussian (a logistic) function.

Software 10.2.2: (Testing half Gaussian as the logistic function)

A program **thgistic.cpp** has been developed and listed in Appendix(5.5). Its outputs are shown in Figure(10.4)



(a) Input cosine signal.

(b) Cosine after detecting peaks and setting other elements equals zero.

(c) Reconstructed cosine.

(d) Input cosine signal.

(e) Cosine after detecting peaks and setting other elements equals zero.

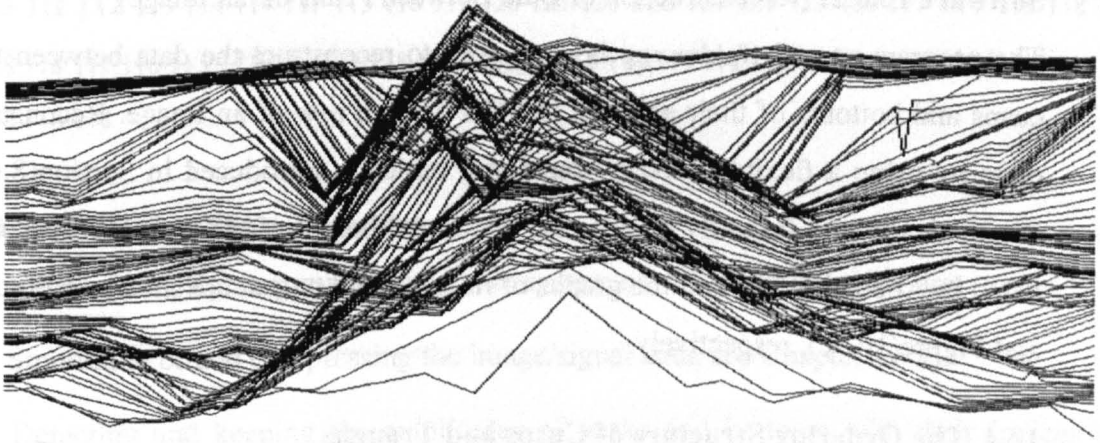
(f) Reconstructed cosine.

Figure 10.4: Outputs of thgistic.cpp to demonstrate the unification of two transforms, the reconstruction and the smoothing, shown in Figure(10.2©,(d)).

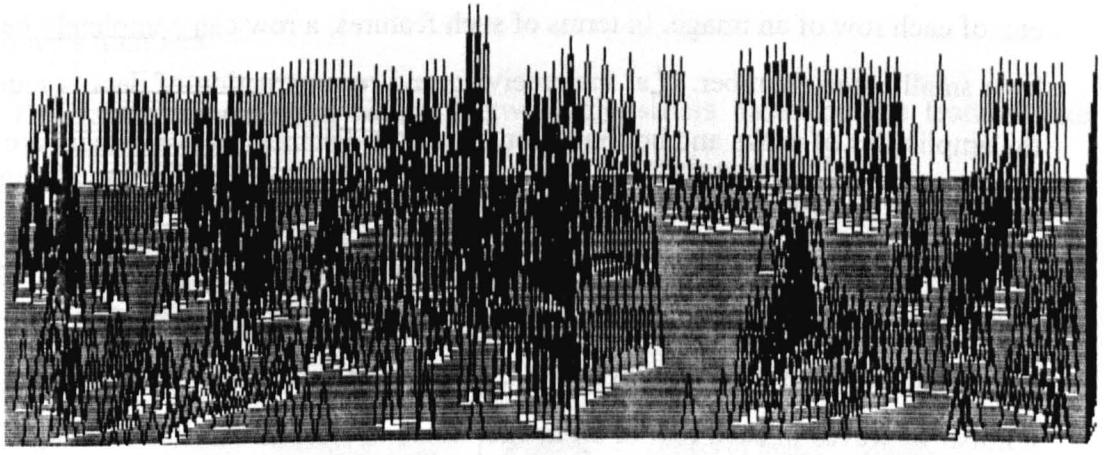
10.3 Peaks Detecting of Images

Software 10.3.1: (Finding peaks of an image and setting other data equals zero)

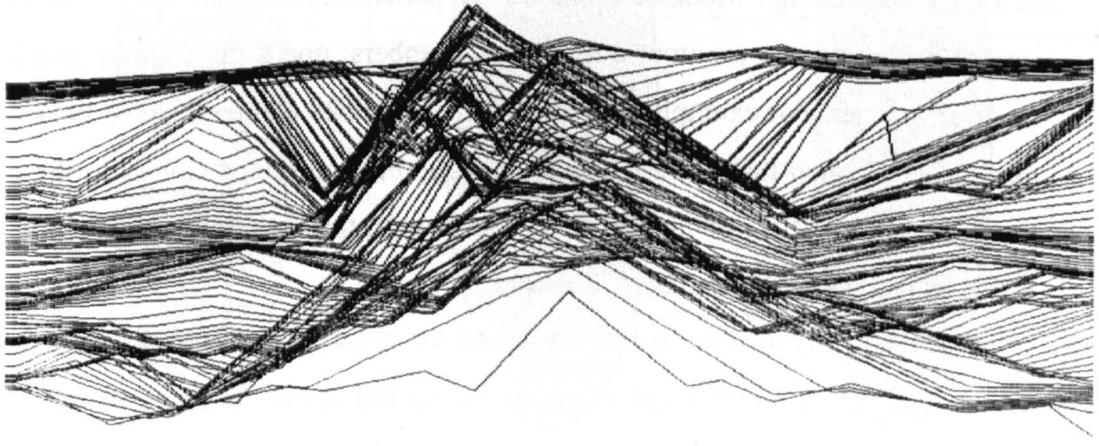
The program named **tfpidz.cpp**is developed to detect the peaks of cusps and bottoms of troughs so that all the other elements be set equals 0 of each row of an image. The program is listed in Appendix(5.6) and the graphs of its input image and output data files are shown in Figure(10.5a) and Figure(10.5b), respectively.



Figure(10.5a): Graph of the image mssmthd.dat which is input to tfpidz.cpp



Figure(10.5b): Graph of the data file tfpidzo.dat which is output from tfpidz.cpp and input to tfpidzr.cpp



Figure(10.5c): Graph of the data file tfpidzo.dat which is output from tfpidzr.cpp

Software 10.3.2: (Reconstruction of Data Between Peaks of an Image)

The program named **tfpidzr.cpp** is developed to reconstruct the data between the peaks cusps and bottoms of their adjacent troughs of each row of an image. Reconstruction is done by using a flexible wavelet transform, basically introduced in Section(8.6), which automatically and instantly decides the width and height of the wavelet. The program is listed in Appendix(5.7) and the graphs of its input and output are shown in Figure(10.5b) and Figure(10.5c), respectively.

10.4 The Ordering Structure of Cusps and Troughs

Data can further be reduced by developing a binary number to represent, the ordering structure of occurrence of cusps and troughs, the number of waves, the widths of waves, etc. of each row of an image. In terms of such features, a row can completely be specified by a small binary number. If at the receiving end reconstruction of data is required then the amplitudes of peaks and bottoms along with the distances between them are sufficient to specify a row and its reconstruction.

Software 10.4.1: (Testing of Wave Count)

A program named **twkount.cpp**, listed in Appendix(5.8), is developed as an example to detect the first and the last wave as a cusps or/and as a trough plus to count the total number of waves in each row of an image. Since in a signal (row of an image) after each cusps there is definitely a trough and vice versa, therefore, in addition to find the number of waves, a flag showing that whether at the beginning and at the end of the signal there is a cusp or a trough becomes a feature for pattern recognition. Corresponding to each row of an image, the program outputs two numbers; one 2-digit binary number and the other an integer. For example, consider the row of 128 elements given below. There are total 6 waves such that there is the cusp in the beginning and trough in the end. So the program will out "10 6" as feature of this row. In this way, an image having dimension 128x128 can be recognised by 128 numbers.

97 97 97 98 99 100 101 101 102 **102** 101 100 99 99 98 98 97 97 95 93 **93** 95 97 98 **99** 98 97
95 94 93 93 **93** 94 94 95 95 96 97 98 100 101 102 **102** 101 99 98 97 96 96 **96** 98 100 103 106
110 113 116 118 121 122 124 125 126 128 130 133 134 **134** 132 129 125 123 123 122 120 118 115 114

113 112 **112** 113 114 114 115 115 116 117 120 123 126 **128** 127 125 123 120 117 **117** 118 118 **119** 118 118 117 116 115 **115** 116 117 118 118 118 118 118 118 118 118 118 **118** 117 117 **116** 117 118 118 118 118 118

10.5 Developing Phase Detectors

The stages to design a complete hardware system of the ANN classifier are:

1. Smoothing or/and compressing the image/signal data, see Chapters(7-9).
2. Detecting and keeping the amplitudes of peaks and bottoms with their indices and setting the other data elements equals 0.
3. Counting the number of waves and specifying the ordering structure of the peaks with binary numbers.
4. Designing the Artificial Neural Network Classifiers based on the features like the number of waves, ordering structure of peaks and their amplitudes.

The Basic Structure of the Image Classifier

The basic structure of an image classifier is shown in Figure (10.5)

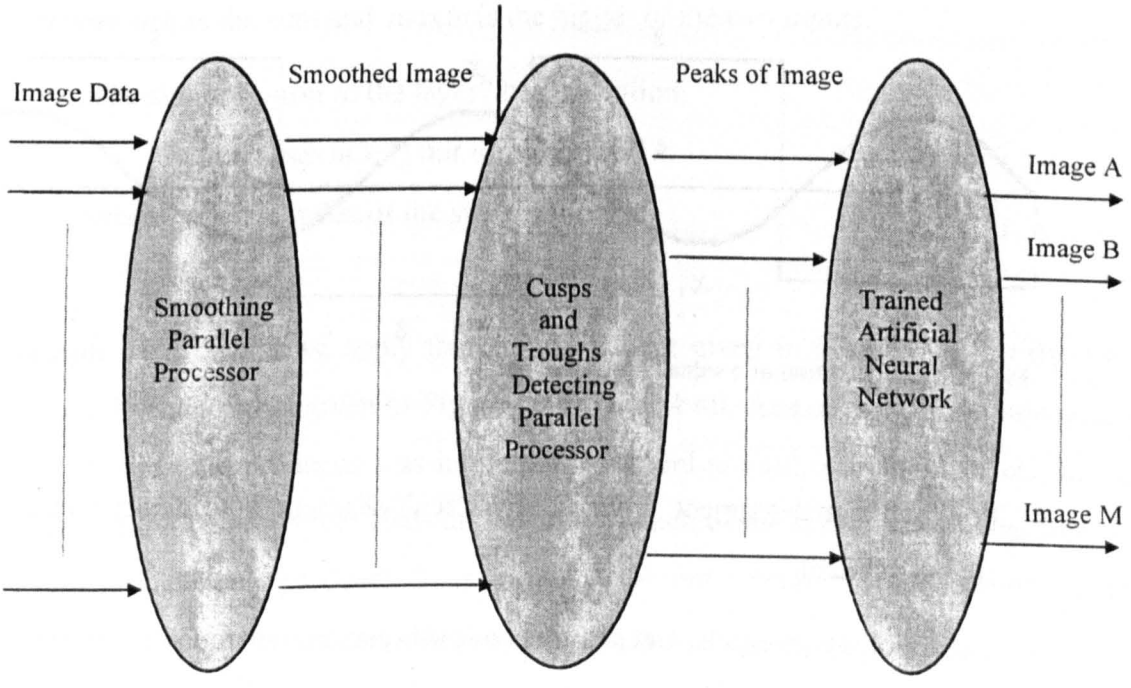


Figure 10.5: Basic Structure of an Image Classifier.

10.6 Data Reduction by Detecting Peaks of Cusps

To minimise the cost and processing time, Parallel Processing and Data Reduction can not be avoided. An appropriate procedure to reduce the data is to detect corner points (cusps and troughs) of a signal. In this section, the task of detecting cusps will be perform by introducing a parallel processor as:

1. It is required to catch the index values of peaks of cusps. Catching bottoms of troughs will be expressed in Section(10.10).

2. All the rows of an image are to be processed simultaneously on parallel lines. A row can be thought as a one-dimensional array of memory units. The Array can be partitioned into segments such that there is at most one cusp or/and one trough in a single segment (the best length equals 4 will be justified in Section(10.9). This is quit possible and practically has been observed.

Let us partition the array into segments, each having 8 elements, as shown in Figure(10.6).

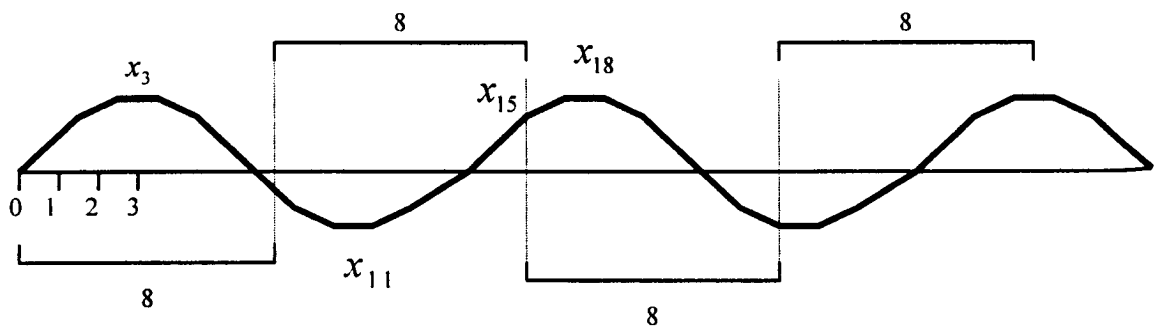


Figure 10.6: Partition of a signal into octive parts

4. For each signal-segment k we need a circuit-segment k to detect a cusp from that signal-segment.

5. The topology , weights, and activities of the layers of the required circuit is explained in the following.

Topology and Weights:

- There are 8 layers in all.
- The j^{th} layer has (8-j) neurons, j=0,1,...,7.
- All connections are forward.
- Weights on all connections are shown in Figure(10.7).

Activation:

- Layer0 is the input layer and does no operation.
- All neurons of layer1 apply the first order difference equation such that

  $$if((x_{i+1} - x_i) \geq 0)\ out = 1 \text{ otherwise } out = 0.$$

- All neurons of layers2 have same activation which is defined as;

  $if(net_i = 1)\ out = 1$ otherwise $if(net_i > 1)\ out = \max in_i$ otherwise $out = 0$

  where w_i and w_{i+1} are the weights, net_i is the weighted sum of inputs, and $\max in_i$ is the bigger of the two weighted inputs of i^{th} neuron.

- All the neurons of layers3-layer6 have the same activation which is defined as;

  $if(net = 1)\ out = 1$ otherwise $if(net > 1)\ out = \max in$ otherwise $out = 0$

  where net is the sum and $\max in$ is the bigger of the two inputs.

- The single neuron of the layer7 has activation;

  $$if(\max in \geq 1)\ out = \max in + 8 * k$$

  where k is the index of the sub circuit.

Example 10.6: When we apply the circuit-segment given in Figure(10.7) to the signal segment 0 of the signal given in Figure(10.6), then k=0. And each of first three neurons of layer1 fires and produces 1 as its output. First and second neurons of layer2 fire and output 2 and 3 respectively. Then first neuron of each of the layer3-layer6 fire, one after the other, and each produces 3 as output. The only neuron of layer7 will output 3+8* k =3; which is the index of the peak of first cusp of the signal.

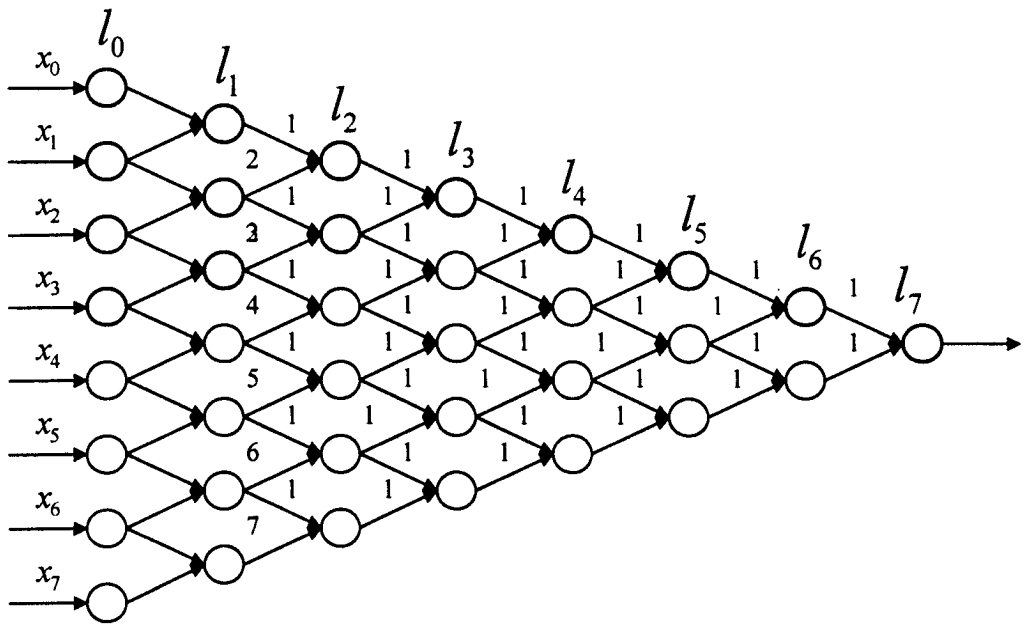


Figure 10.7: A circuit segment to detect index of the peak of a cusp.

10.7 Optimality for Data Reduction

There may be a case when a single peak is detected twice; once in each of the two consecutive segments. For example, in second and third segment of Figure(10.8) a single peak is detected twice; once at the point x_{15} and then at x_{18}. This may be harmless for pattern recognition but circuit-size can be reduced if we detect and unify such peaks by designing a circuit as shown in Figure(10.8). In order to intimate about the happening of the case, there are two neurons connecting the adjacent segments of the circuit and have activation given below:

Non-shaded neuron: Its activation is the same as that of neurons of layer1 expressed in Section(10.6).

Shaded neuron:

$$if\ y_1 + y = 2\ \text{ or }\ y + y_2 = 2\ \text{ then } out = 0 \text{ otherwise } out = 1$$

where y_1, y, and y_2 are the inputs of the neuron coming from the preceding segment, the non-shaded neuron, and the proceeding segment respectively. Obviously the output of the neuron will be 0 when the actual peak will be going to next circuit-segment.

The new output neuron of each circuit-segment always outputs the product of its inputs. Hence it is clear that the small shaded neuron causes to keep silent the preceding circuit-segment when the actual peak goes to the next circuit-segment. This is equivalent to that

as there were no peak detected by the preceding circuit-segment. For example, the index of the element x_{15} of the signal shown in Figure(10.6) will be detected as a peak by the second circuit-segment. But its declaration will be prevented by the small shaded neuron because the actual peak goes to the next circuit-segment.

10.8 Optimal Circuit Explanation

Let us process the signal shown in Figure(10.6) by applying the circuit shown in Figure(10.8). The signal segment k will be processed by circuit segment k, $k = 0,1,...,n/8$; where n is the length of the signal. In this particular case, darker neurons fire while the others will remain silent.

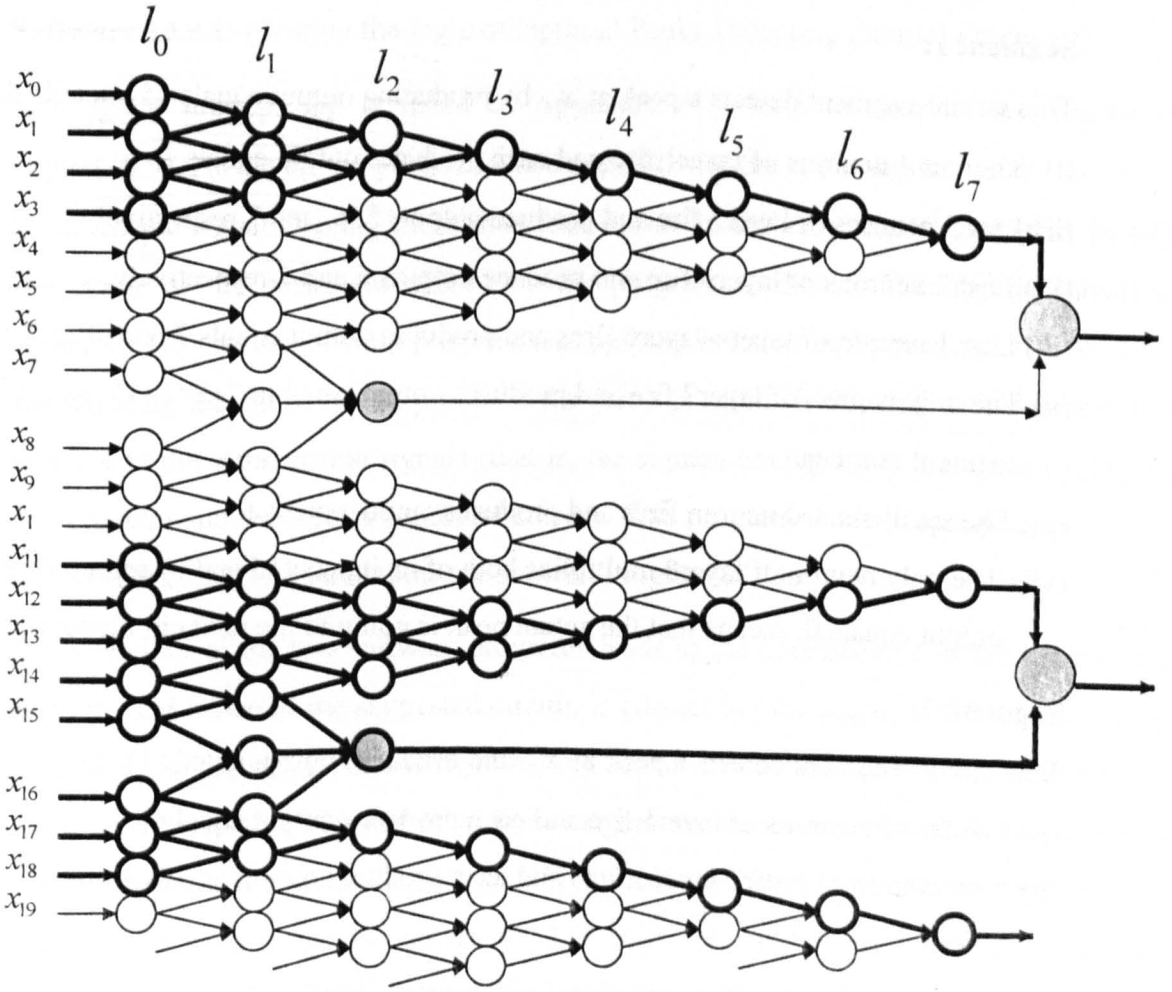


Figure 10.8: Optimal Peaks Detecting Parallel Processor.

Segment 0:

This circuit-segment detects a peak at x_3 by producing output equals 3. Details follow;

(i) Layer0 does no operation. It passes its inputs directly to layer1.

(ii) First 3 neurons of layer1 fire and each produces output equals 1.

(iii) First 2 neurons of layer2 fire and produce outputs equals 2 and 3 respectively.

(iv) Only first neuron of layer3-layer6 fire and produces output equals 3 one after the other..

(v) The only neuron of layer7 fires and produces output equals 3+8*0=3, where 0 is the segment number.

(vi) The small shaded neuron do not fire and therefore its output will be thought of 1.

(vii) The only neuron of layer8 multiplies its inputs (3 and 1) and produces output 3.

Segment 1:

This circuit-segment detects a peak at x_{15} by producing output equals 15. Details follow;

(i) The last 4 neurons of layer1 fire and each produces output equals 1

(ii) Last 3 neurons of layer2 fire and produce outputs 5, 6, and 7 respectively.

(iii) Last 2 neurons of layer3 fire and produce outputs 6 and 7 respectively.

(iv) Last 1 neuron of layer4-layer6 fires and produces output equals 7 one after the other.

(v) The only neuron of layer7 fire and produces output equal to 7+8*1=15, where 1 is the segment number.

(vi) The small shaded neuron fires and produces output equals 0.

(vii) The only neuron of layer8 multiplies both of its inputs (15 and 0) and produces output equals 0; means that the actual peak is going to the next circuit-segment.

Segment 2:

This circuit-segment detects a peak at x_{18} and produces output equals 18. Details follow;

(i) The first 2 neurons of layer1 fire and each produces output equals 1.

(ii) First neuron of layer2-layer6 fires and each produces output equals 2 one after the other.

(iii) The only neuron of layer7 fires and outputs 2+8*2=18, where multiple 2 is the segment number.

(iv) The small shaded neuron do not fire and therefore its output is thought of 1.

(v) The only neuron of layer8 multiplies its inputs (18 and 1) and produces output 18.

Thus the circuit has detected 2 cusps, at x_3 and x_{18}, of the signal through its 3 segments.

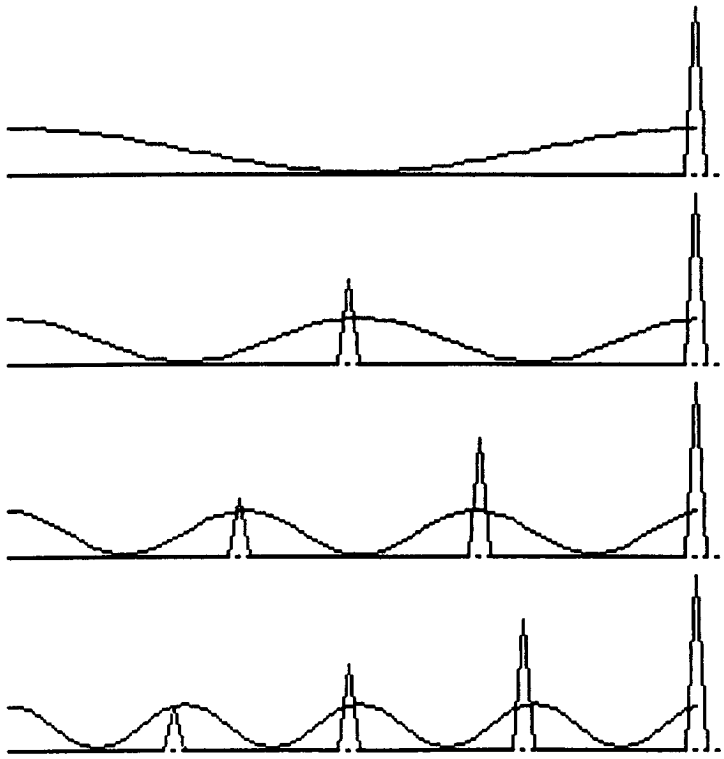
10.9 Checking logic of the Peaks Detecting Parallel Processor against Software

There are total three software programs which are developed to test and demonstrate the logic of the suggested Optimal Peaks Detecting Parallel Processor shown in Figure(10.8). While a fourth program which is for bottom detection will be presented in Section(10.10). The code of all the programs clearly demonstrates the design topology and the logic of he processor explained in Section(10.6)-Section(10.8).

Software 10.9.1: (Testing the logic of Optimal Peaks Detecting Parallel Processor)

A program **topdpp1.cpp** is listed in Appendix (5.9) and its outputs are shown in Figure(10.9). However. the program when applied with the same segment-size (that is, 8) to the cosine signals having more than four cycles in the same signal size (that is, 64), then some of the peaks are missed by the program to detect. This fact is shown in Figure(10.10) and its software program **topdpp2.cpp** is listed in Appendix(5.10). Without investigating the cause of the problem, the same program is applied with segment-size equals 4 to the same cosine signals (that is, the signals having more than four cycles per 64 elements) and the results are found correct. This program named **topdpp3.cpp** is listed in Appendix(5.11) and its outputs are shown in Figure(10.11).

The behaviour of the two software programs leads to the conclusion that the logic of the program, and hence of the suggested circuit, is correct but the segment size equals 8 may not be helpful for signals and images having higher frequency contents. It is also a fact that three consecutive element of a digital signal are necessary and sufficient to check whether the middle point is the peak or a bottom, if any. Moreover, we are interested in a segment-size equals integral power of 2. Hence the only valid option is to choose segment-size equals 4. Thus a modified circuit which is shown in Figure(10.13) is suggested.



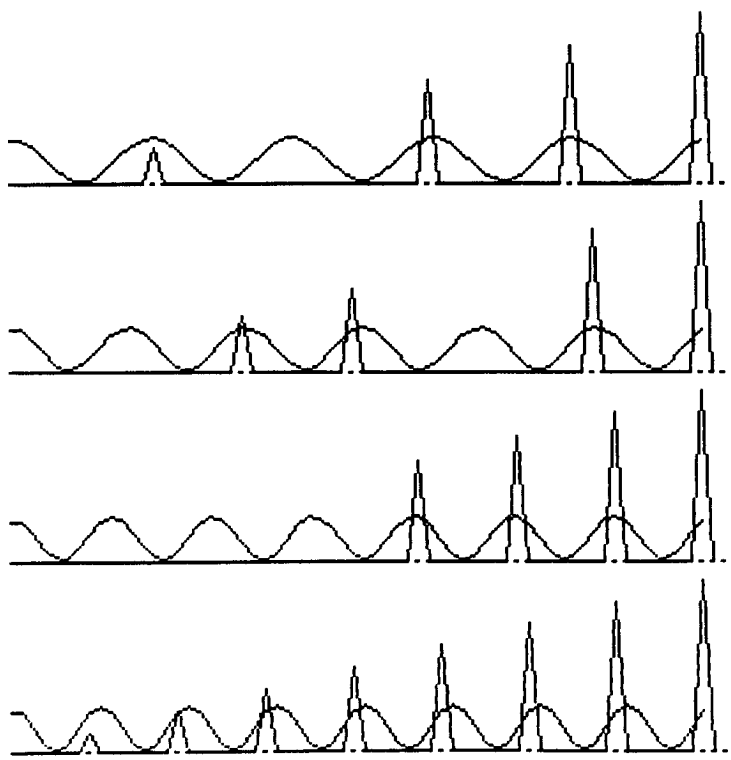
(a) Input cosine signal with its peak detected.

(b) Input cosine with its two peaks detected. of the peak detected.

© Input cosine with its three peaks detected. (a) .

(d) Input cosine with its four detected. of the peak detected.

Figure 10.9: Outputs of topdpp1.cpp to test the logic of optimal peaks detecting parallel processor shown in Figure(10.8).



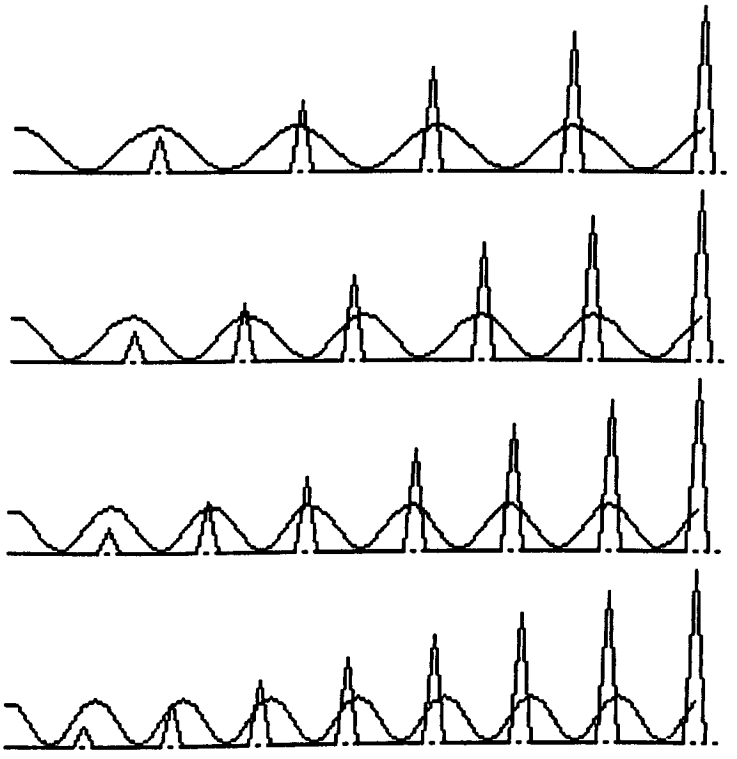
(b) Outputs showing that the second peak from left to right is missed.

(b) Outputs showing that the first and the fourth peak is missing. and fourth peaks are missed.

© Outputs showing that the first three peaks are missing.

(a) Result in this case is OK.

Figure 10.10: Outputs of topdpp2.cpp to demonstrate the limitations of the logic circuit shown in Figure(10.8).



In each of these four cases, the peaks of the cosine signals have been detected correctly by the program.

Figure 10.11: Outputs of topdpp3.cpp to the logic of peaks detecting parallel processor by using segment-size equals 4.

10.10 Data Reduction by Detecting Troughs

The procedure adopted for detecting peaks of cusps is equally good for detecting bottoms of troughs, with the only modification:

That the activation of all the neurons of layer1 (including the circuit-segment connecting neuron existing below to layer1 in Figure(10.8)) should be changed from

$$l_1 : if((x_{i+1} - x_i) \geq 0)\ out = 1 \ \text{ otherwise } \ out = 0$$

to

$$l_1 : if((x_{i+1} - x_i) \leq 0)\ out = 1 \ \text{ otherwise } \ out = 0$$

Software10.10: (Testing the logic of Optimal Bottoms Detecting Parallel Processor)

A program **tobdpp.cppis** listed in Appendix(5.12) and its outputs are shown in Figure(10.12). The program is developed with segment-size equals 4. The code of the programs clearly reflects the topology of the processor and the activation of each neuron. Reading the program is therefore suggested in order to understand the structure of the processor.

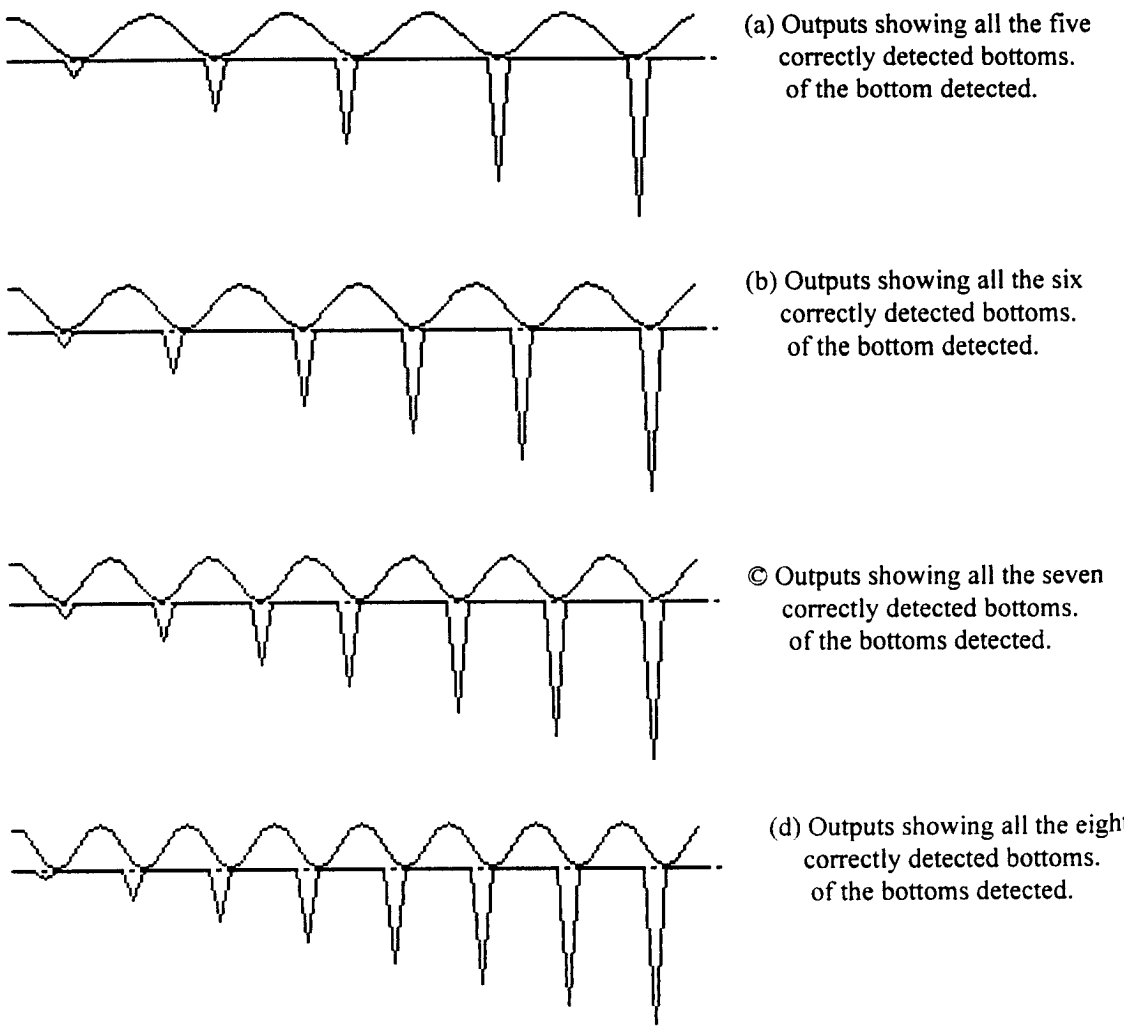


Figure 10.12: Outputs of tobdpp.cpp to test the logic of the suggested optimal bottoms detecting parallel processor.

10.11 A Modified and Finalised Peak Detecting Parallel Processor with Software Check

The circuit of this processor is shown in Figure(10.13) and its performance can be seen in Figure(10.11) which shows the outputs of **topdpp3.cpp** listed in Appendix(5.9).

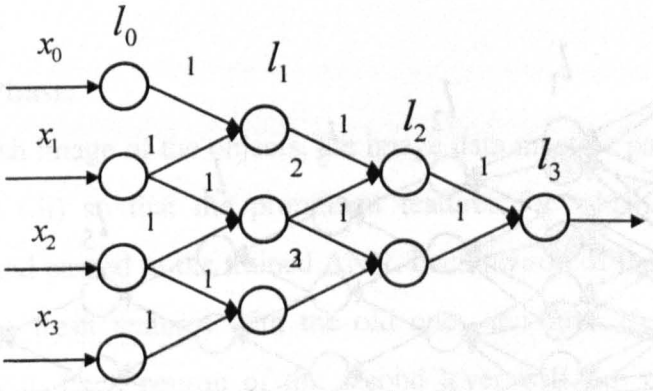


Figure 10.13a: A peak detecting circuit-segment to show weights.

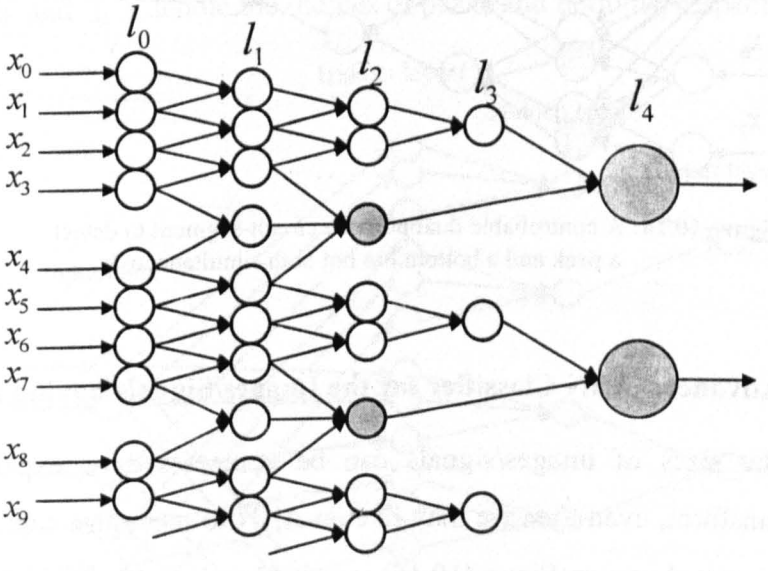


Figure 10.13b: A Modified Optimal Peaks Detecting Parallel Processor.

10.12 An Advanced Dual Circuit for Peaks and Bottoms Detecting ANN

Both the tasks of detecting peaks and bottoms can be performed by a single processor if we design its circuit-segments as shown in Figure(10.14). The single black neuron, can be called a controller, outputs a single bit control code to enable one, and at the same time

disable the other with the same code, of two options of layer1. If the shaded neurons of layer1 are reserved for bottoms then the non-shaded neurons will have to be reserved for peaks and vice versa. A complete parallel processor can be design by connecting these circuit-segments on the same line as were adopted in Section(10.8) and Section(10.11). A circuit for segment-size equals 4 can designed in a similar way.

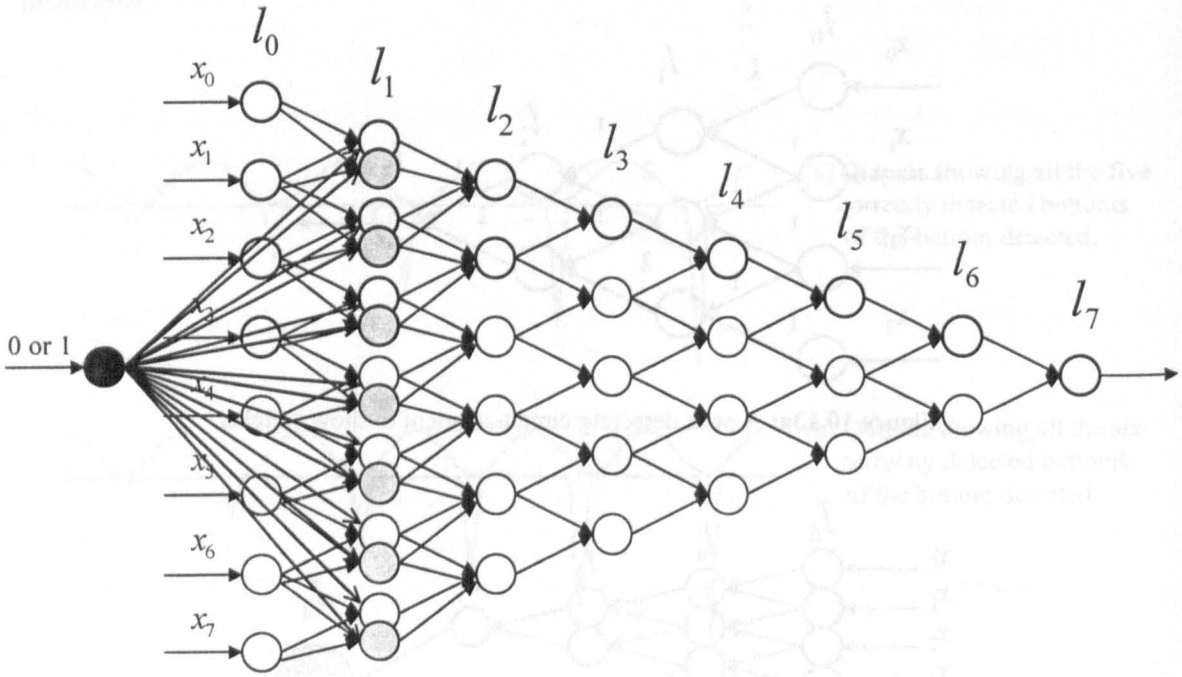


Figure 10.14: A controllable dual purpose circuit-segment to detect a peak and a bottom but not both simultaneously.

10.13 An Advanced ANN Classifier for the Images/Signals having Fixed Sizes

Although the sizes of images/signals can be compressed or expanded by applying Wavelet Transform, even then we may like an ANN to recognise signals/images of fixed sizes. The circuit shown in Figure(10.15) is suitable when the sizes of the images/signals remains or kept the same during both the training and the working phase.

Training Phase:

Collect images of the objects A through M and follow the training steps given below:

(i) Remove noise and compress/expand the images by using some ANN described in Chapter(7-9);

(ii) Locate peaks and bottoms in each row of the images by using an ANN as given in Figure(10.13b);

(iii) Compute some prominent features like the number of, the amplitudes, and the relative distances of the peaks and bottoms;

(iv) Save the amplitudes of peaks/bottoms and the prominent features in first hidden layer;

Working Phase:

For any fresh image of the objects, the image data must be passed from the training steps (i) through (iii) so that the prominent features for which the ANN was trained be computed and passed to the trained ANN. Each neuron of the first hidden layer will then compare the fresh features with the old ones and fires if it finds a match otherwise remains silent. Each neuron of the second layer will fire if all of its inputs are high otherwise remains silent. An output neurone will fire if both of its inputs are high. The inputs c_i's and T_i's denote the indices of peaks and bottoms, respectively.

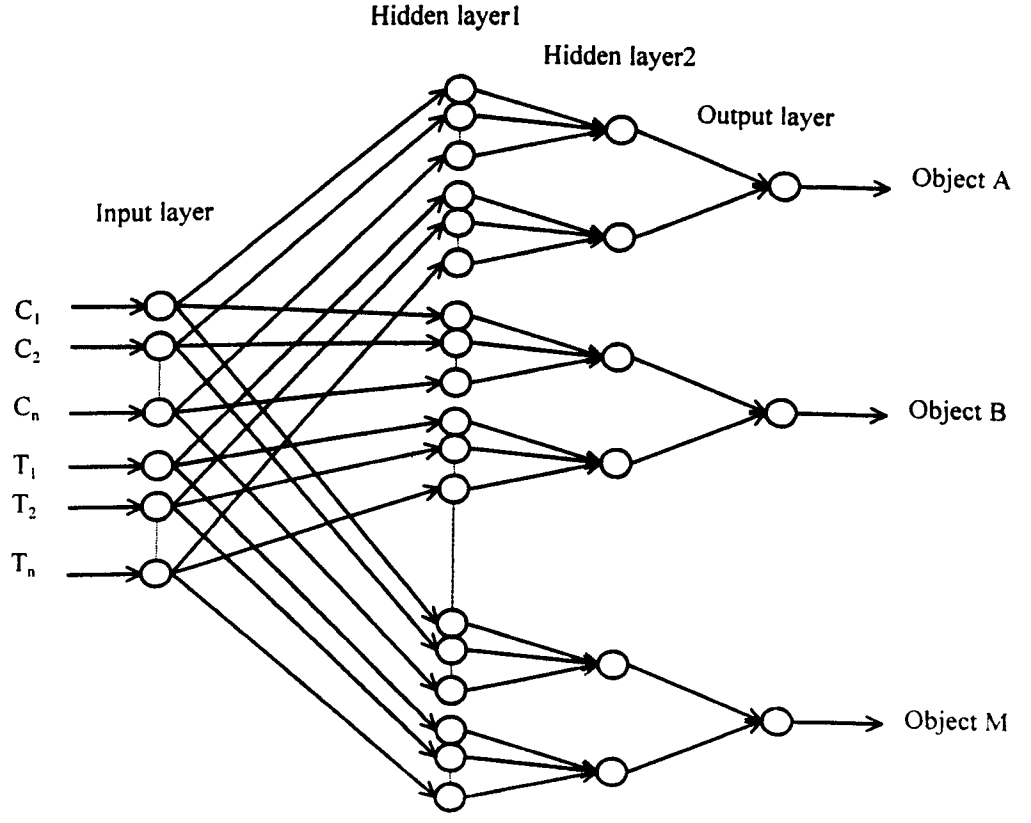


Figure 10.15: ANN Classifier for Images/Signals having Fixed Sizes.

10.14 An Advanced ANN Classifier for Telescopic Images/Signals

The circuit of the ANN shown in Figure(10.16) is able to detect and report a smaller and a larger version of the image/signal used during training of the ANN.

Training: Its training is the same as given in Section(10.13).

Activation of hidden layer1: Each neurone computes the fresh distance $fd = c_{i+1} - c_i$ and divides it with the distance $td = c_{i+1} - c_i$ computed in the training phase. It fires and its output equals -1, or 0, or 1 if the ratio $\frac{fd}{td}$ is less than, or equals, or greater than 0.

Activation of hidden layer2: A shaded (or a hollow or a solid) neuron of the layer fires if all of its inputs equals -1 (or 0, or 1, respectively).

Activation of output layer: An output neuron fires if both of its inputs are high.

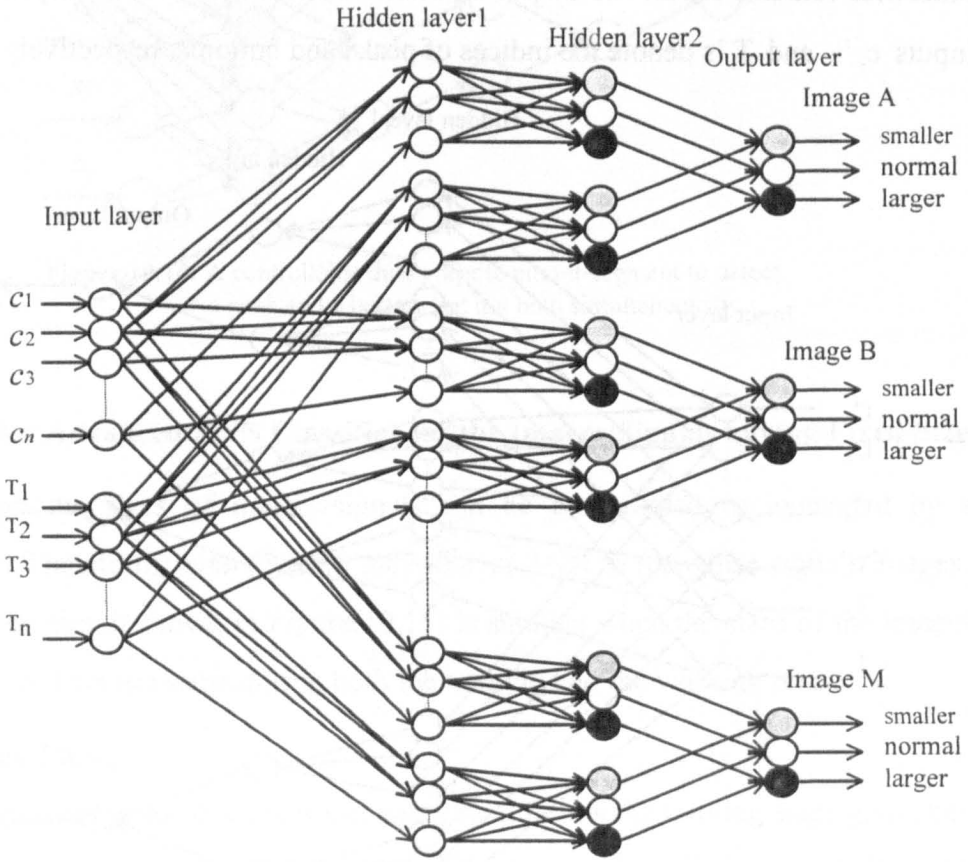


Figure 10.16: ANN Classifier for Telescopic Images/Signals.

10.15 An Advanced ANN Classifier Based on the Ordering Structure

It has been described in Section(10.4) that each row of an image can be recognised by associating a binary number with it. If we slightly modify the logic and some parameters of Section (10.6) and Section(10.11), then the processors shown in Figure(10.8) and Figure(10.13b) become suitable to find the binary number. However, the Artificial Neural Network Classifier based upon this technique is shown in Figure(10.17). Each neuron of first hidden layer will compare the fresh binary nubmer with the binary number inherited to it during training and fires if it finds a match. An output neuron will fire if all of its inputs are high. This ANN can be modified to fit for the logic presented in **twkount.cpp** described in Section(10.4).

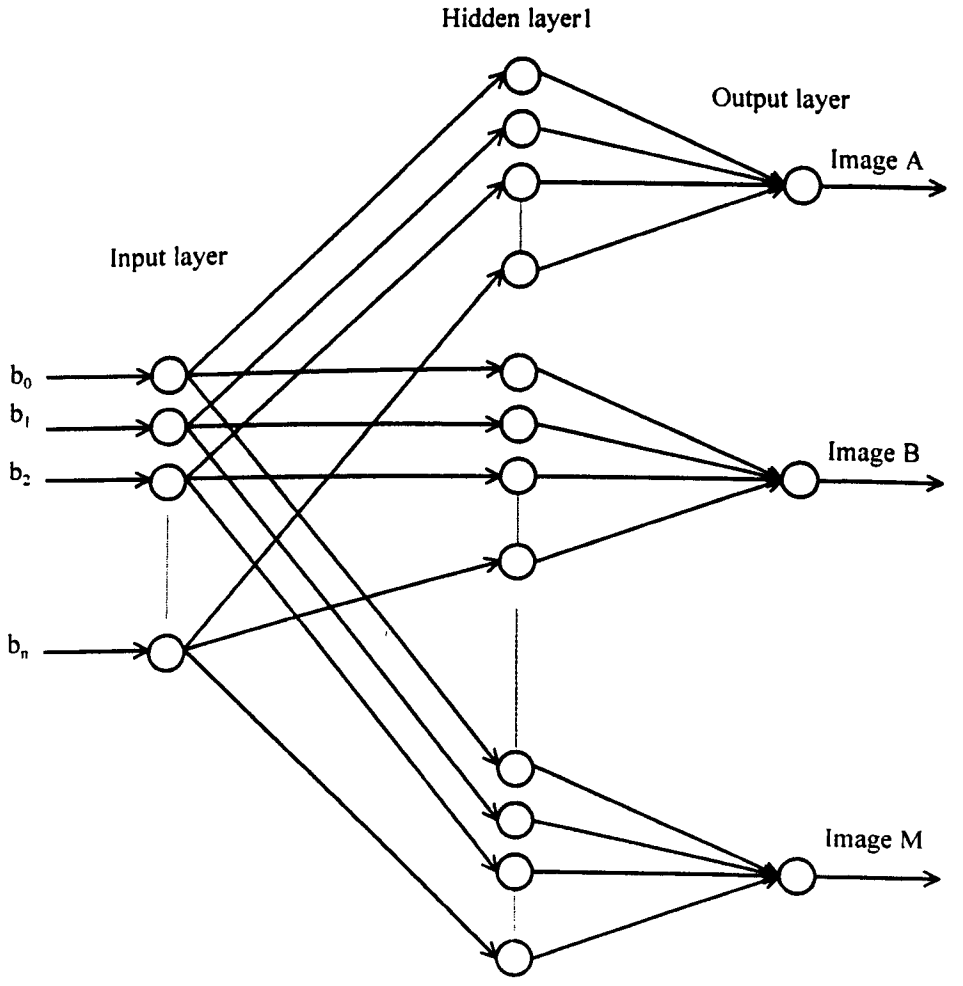


Figure 10.17: ANN Classifier based upon the Ordering Structures.

Design Logic of Bit-wise Parallel Binary Adder Circuits

From the previous chapters it is clear that millions of similar processors are required almost in every ANN. Consider, for example, the two-dimensional data smoothing wavelet ANN shown in Figure(9.3.1), where the wavelet

$\frac{1}{10}$

1	1	1
1	2	1
1	1	1

has been used. There, each neuron of output layer performs first the direct multiplication of nine elements of the wavelet with the image elements which are under the window and then sums up the resulting products. Since there are millions of window placing, hence the millions of output neurons are required to perform millions of the weighted sums.

Is this a wise step to use millions of the existing micro processors which are sequential as well?; Can we afford their price?; Can we appreciate the big sizes of the resulting components?. Certainly No, because we are able to design some special purpose small parallel processors. Designing such processors is the issue of this Chapter.

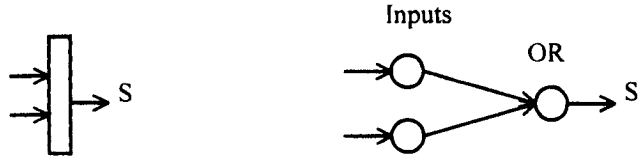
It should be noted that the processors to be presented in this chapter will perform addition of integers and multiplication/division of integral powers of 2.

11.1 Different Units for Bit-wise Parallel Adder Circuits

The following units are required for designing a small parallel processor such as shown in Figure(11.9).

11.1.1 A 2-to-1 adder

Although logical OR gate is very basic but its symbolic reference is required in Figure(11.9). Its symbol, logical circuit and truth table are shown in Figure(11.1).

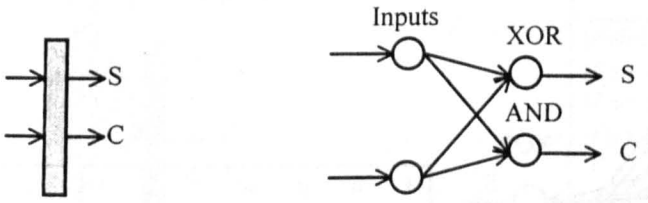


Inputs		S
0	0	0
0	1	1
1	0	1

Figure 11.1: A 2-to-1 Adder, its circuit, and truth table

11.1.2 A 2-to-2 adder

Some times we need to add two bits whether both are high or not. The symbolic representation, the logical circuit and the truth table of 2-to-2 adder is shown in Figure(11.2).

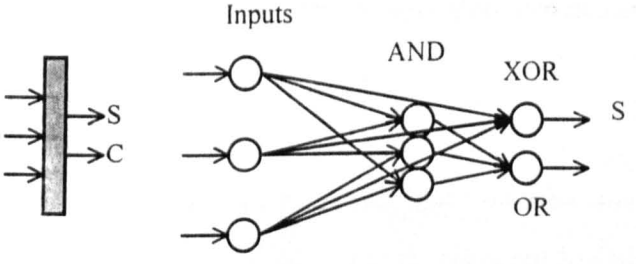


Inputs		C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figure 11.2: A 2-to-2 Adder, its logical circuit, and truth table

11.1.3 A 3-to-2 adder

The symbol, the logical circuit and the truth table of 3-to-2 adder is shown in Figure(11.3).

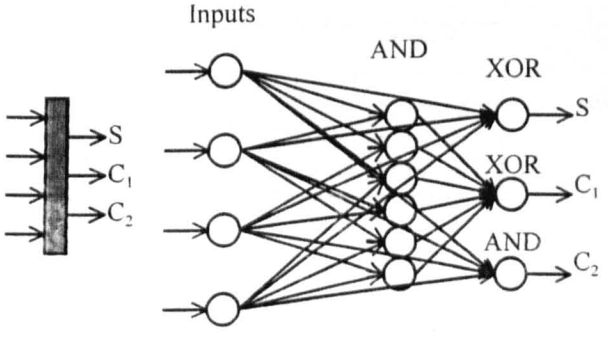


Inputs			C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
1	0	0	0	1
0	1	1	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figure 1.3: A 3-to-2 Adder, its logical circuit, and truth table

11.1.4 A 4-to-3 adder

The symbol, the logical circuit and the truth table of 4-to-3 adder is shown in Figure(11.4).

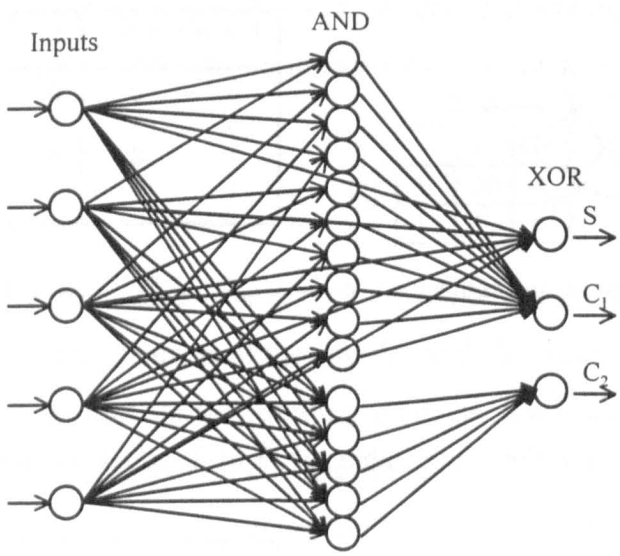


Inputs	C_2	C_1	S
If all inputs are low	0	0	0
else if no. of high inputs is one	0	0	1
else if no. of high inputs are two	0	1	0
else if no. of high inputs are three	0	1	1
else if no. of high inputs are four	1	1	0

Figure 11.4: A 4-to-3 bit Adder, its circuit, and truth table

11.1.5 A 5-to-3 adder circuit

The symbol, the logical circuit and the truth table of 5-to-3 adder is shown in Figure(11.5).

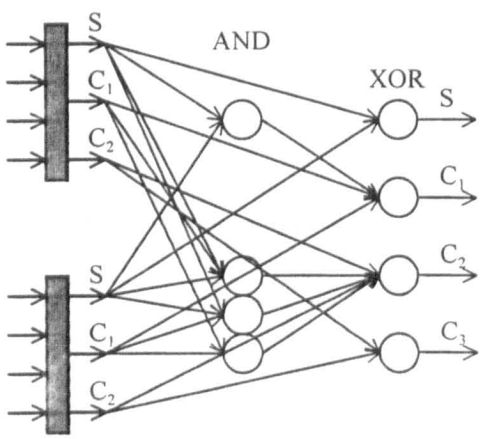


Inputs	C_2	C_1	S
Initially	0	0	0
If no. of inputs are odd			1
If no. of high couples is odd		1	
If no. of high 4-touples is odd	1		

Figure 11.5: A 5-to-3 bit Adder circuit and its truth table

11.1.6 An 8-to-4 adder

The logical circuit and the truth table of 8-to-4 adder is shown in Figure(11.6). Two 4-to-3 adders are used here for optimality of the logic circuit.

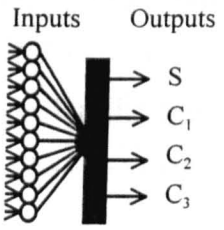


Inputs	C_3	C_2	C_1	S
Initially 0 0	0	0		
if no. of high inputs is odd				1
if no. of high couples is odd			1	
if no. of high 4-touples is odd		1		
if no. of high 8-touples is odd	1			

Figure 11.6: An 8-to-4 adder circuit.

11.1.7 A 10-to-4 adder circuit

Basic structure of the adder is shown in Figure(11.7).



I n p u t s	C_3	C_2	C_1	S
Initially	0	0	0	0
If no. of inputs are odd				1
If no. of high couples is odd			1	
If no. of high 4-touples is odd		1		
If no. of high 8-touples is odd	1			

Figure 11.7: Basic structure of 10-to4 bit adder and its truth table

The basic structure given in Figure(11.7) is expressed in details in Figure(11.8). Where there are two (strips of) 5-to-3 adders that were introduced in Figure(11.5) and here are shown in the form of strips. These strips together with the part of the circuit that follows to the strips constitute the required 10-to-4 adder circuit.

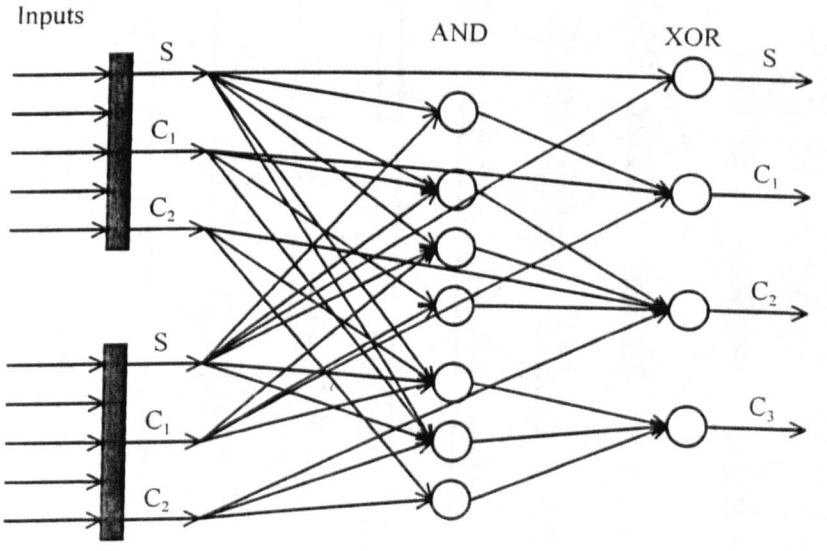


Figure 11.8: Detailed 10-to-4 bit adder circuit

A 9-to-4 adder circuit: A 9-to-4 adder can be achieved by replacing one 5-to-4 strip in Figure(11.8) by a 4-to-3 strip.

11.2 Designing Bit-wise Parallel Adder Circuit to add nine integers

The following Parallel Adder Circuit is developed to add at most nine 8-bit integers.

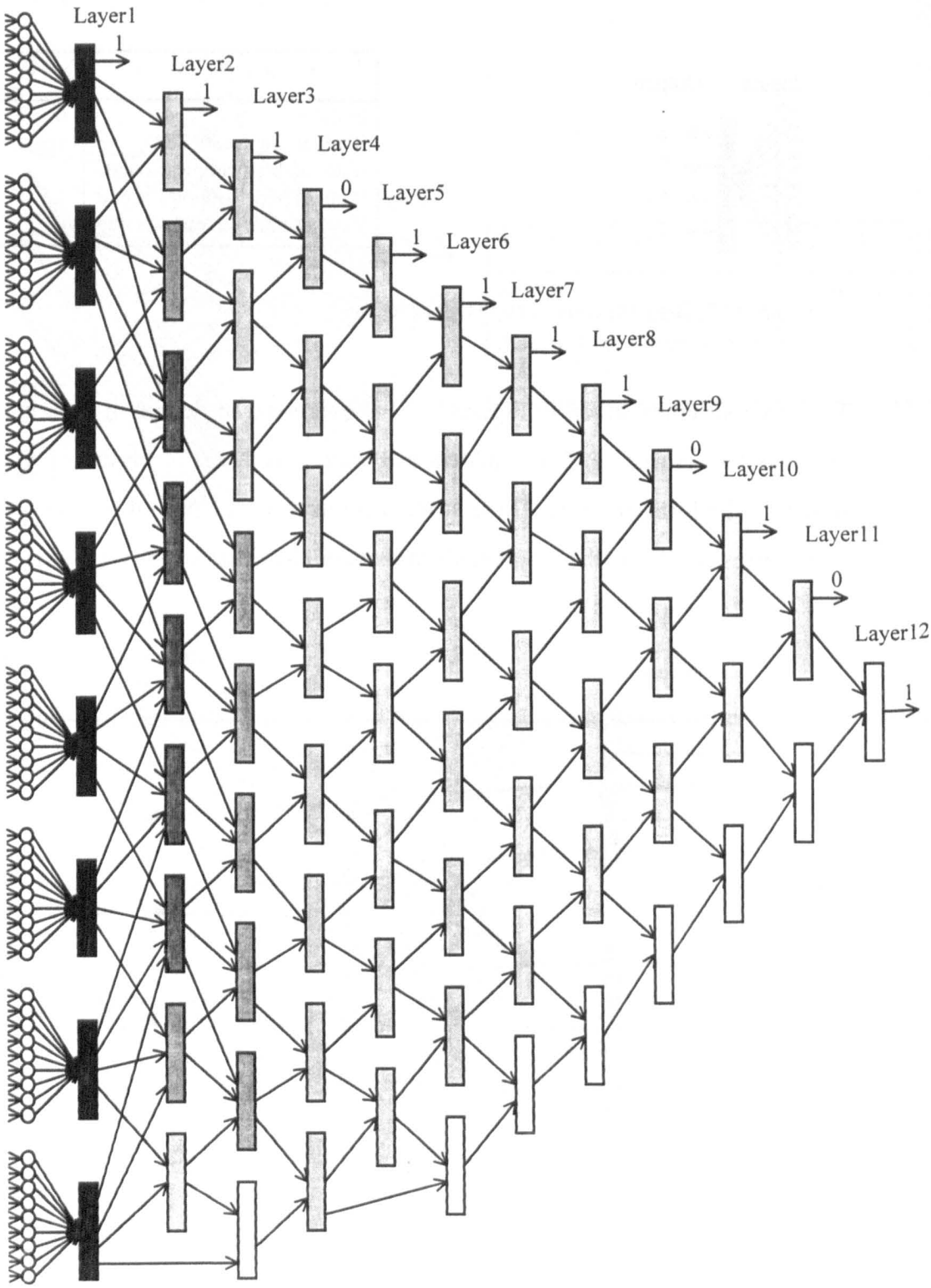


Figure 11.9: Parallel Adder Circuit to add at most nine 8-bit integers.

11.3 Description of the Parallel Adder Circuit Shown in Figure(11.9)

The image data is stored in binary form in contiguous memory locations and bit by bit connections are made to an adder circuit. Consider, for example, nine integers each having binary value equals 11111111. Write the integers in the form as shown in Table(3.1). The bits of individual columns are added according to the decimal number system but the sum be written in binary form as shown in Table(3-1) through Table(3-12). The layer by layer processing of the circuit follows.

11.3.1 Layer 1:

Write the given 9 integers in the form of Table(3-1). Bits of a column of the table are added by a unit of layer1 of the Adder Circuit shown in Figure(11.9).

Table 3-1

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1001	1001	1001	1001	1001	1001	1001	1001

All the units of the layer are same and there are 9 inputs to each unit; 4 outputs from each unit; from top to down the unit number k adds the 2^k th bits of the 9 integers, where $k = 0,1,...,7$.

11.3.2 Layer 2:

Write the results of Table(3-1) in the form of Table(3-2). Bits of each column of the Table(3-2) are added by a corresponding unit of the layer2. The boxed 1 in the table is the 2^0 bit of ultimate result.

Table 3-2

									1	0	0	1			
								1	0	0	1				
							1	0	0	1					
						1	0	0	1						
					1	0	0	1							
				1	0	0	1								
			1	0	0	1									
		1	0	0	1										
1	01	01	010	010	010	010	010	01	01	[1]					

11.3.3 Layer 3:

Results of Table(3-2) are added by the layer3 as shown in Table(3-3). The boxed 1 is the 2^1 bit of ultimate result.

Table 3-3

												0	1		
											0	1			
									0	1	0				
								0	1	0					
							0	1	0						
						0	1	0							
					0	1	0								
				0	1	0									
			0	1											
		0	1												
	0	1													
1	01	10	01	01	01	01	00	01	[1]						

11.3.4 Layer 4:

Results of Table(3-3) are added by the layer4 as shown in Table(3-4). The boxed 1 is the 2^2 bit of ultimate result.

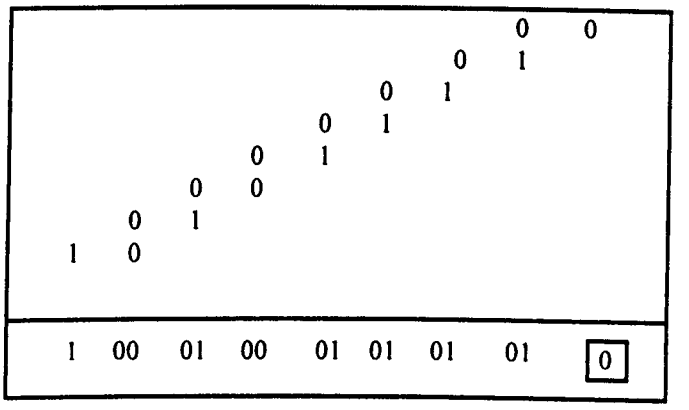
Table 3-4

												0	1		
											0	0			
									0	1					
								0	1						
							0	1							
						0	1								
					1	0									
				1	0										
			1	0											
10	01	00	01	01	01	01	00	[1]							

11.3.5 Layer 5:

Results of Table(3-4) are added by the layer5 as shown in Table(3-5). The boxed 1is the 2^3 bit of ultimate result.

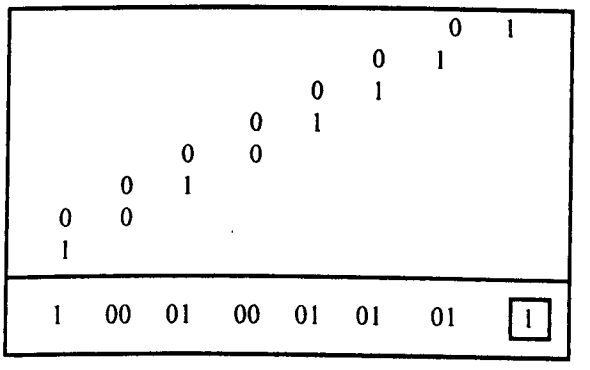
Table 3-5



11.3.6 Layer 6:

Results of Table(3-5) are added by the layer6 as shown in Table(3-6). The boxed 1 is the 2^4 bit of ultimate result.

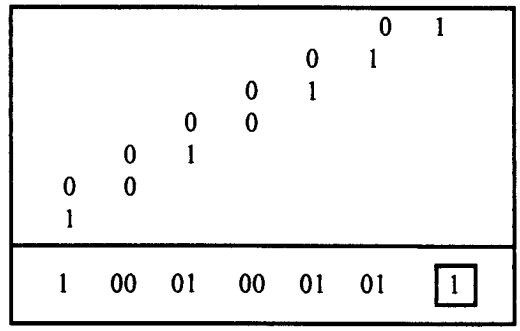
Table 3-6



11.3.7 Layer 7:

Results of Table(3-6) are added by the layer7 as shown in Table(3-7). The boxed 1 is the 2^5 bit of ultimate result.

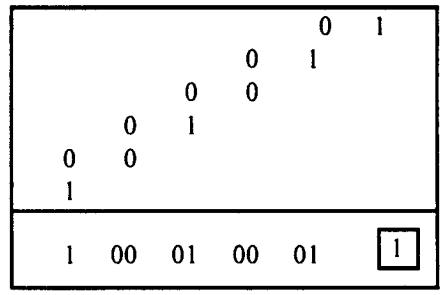
Table 3-7



11.3.8 Layer 8:

Results of Table(3-7) are added by the layer8 as shown in Table(3-8). The boxed 1 is the 2^6 bit of ultimate result.

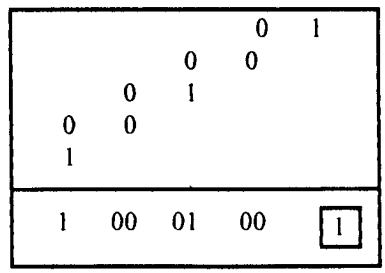
Table 3-8



11.3.9 Layer 9:

Results of Table(3-8) are added by the layer9 as shown in Table(3-9). The boxed 0 is the 2^7 bit of ultimate result.

Table 3-9



11.3.10 Layer 10:

Results of Table(3-9) are added by the layer10 as shown in Table(3-10). The boxed 1 is the 2^8 bit of ultimate result.

Table 3-10

		0	0
	0	1	
0	0		
1			
1	00	01	[0]

5.3.11 Layer 11:

Results of Table(3-10) are added by the layer12 as shown in Table(3-11). The boxed 0 and 1 are the 2^9 and 2^{10} bits of ultimate result.

Table 3-11

	0	1
0	0	
1		
1	00	[1]

5.3.12 Layer 12

Results of Table(3-11) are added by the layer12 as shown in Table(3-12). The boxed 0 and 1 are the 2^9 and 2^{10} bits of ultimate result.

Table 3-12

0	0
1	
[1]	[0]

CONCLUSIONS

C.1 A Drawback of Existing Theory of Networks

Existing theory of Artificial Neural Networks (ANNs) has a drawback due to iterative error minimisation methods for weights training. The ANNs trained by these methods are unable for time-incremental learning. It is proved, by presenting a theorem, in Section(2.1.1) that an error minimisation training method leads to the contradiction to reality that "there exists no two classes separable by their boundaries". Suggestions to overcome the drawback are given at the end of Section(2.1). Readers are suggested to study this section.

C.2 Importance of First hidden Layer

The drawback of error minimisation training methods addressed in Section(2.1) can be overcome if there is a boundary check in the ANN. A proper check can be done by reserving first hidden layer for this purpose. For example, see the design topology of the ANNs given in Figure(4.7) first and Figure(2.11.2) then.

C.3 Weights produced by Delta Rule are Scale Versions of Samples

It is proved in Section(2.2) that the weights successfully produced by Windrow-Hoff delta rule are nothing more than a scaled version of their respective components of the sample pattern. A meaningful scaling can be done by developing an exact method. For example, see the three methods presented in Section(2.3).

C.4 A new Weights Construction Method

A weights construction method is developed in Section(2.3) which is new and consists of a system of n linear equations. The weights enable a network for straight line activation, for the high force and the low force individual thresholding, the high force and the low force segment thresholding, and time incremental learning etc. The weights enable the network to reconstruct and to forecast the missing parts of an incomplete known pattern; even a couple of elements are sufficient for this purpose.

C.5 Importance of the Straight line Activation

- Instead of activating upon individual limits of a feature separately, as is done for the Neuromuscular ANN shown in Figure(4.7), segmentation of signals is suggested so that the high force and the low force threshold straight lines be acted upon.
- An efficient detection of relationships of similarity, between sample patterns and actual patterns, requires segmentation of signals and straight line activation.
- Images can advantageously be classified by extracting the outer boundaries of the objects in the scene. Straight line activation is quit natural for such classification.

C.6 Linear Activation and Linearly Inseparable Classes of Objects

Existing theory of ANNs could not provide some exact method of weights training. It is assessed that developer of the theory could have not recalled or combined the properties of a straight line, the concept of consecutive two-dimensional projections of an n-dimensional sample pattern, and the concept of the greatest lower bound and the least upper bound of a class of patterns. Readers are, therefore, suggested to study Section(2.4) for properties of a straight line and Section(2.5) for the concept of a class of objects and the high force and the low force thresholding boundaries of a class.

C.7 Dividing Boundary of an Image into two Contours

It is explained in Section(2.5) that an object can be classified from the outer boundary of its image data, But in order to classify a class of similar objects, we need to find the high force and the low force threshold boundaries of the class. For this purpose, we need first to find and divide the boundary of each object of the class into two contours. Reader are suggested to study Section(2.6) where a procedure of this task is presented.

C.8 Finding the threshold boundaries of a Class of Images

The concept of a class of the images is explained in Section(2.5). However, classification of the classes requires threshold boundaries of each class. A procedure of finding the threshold boundaries is developed in Section(2.7).

C.9 Linear Activation and Classification of Classes

Section(2.8) is important being a connector of the four developments; the idea of considering successive two-dimensional perpendicular projections of an n-dimensional pattern X expressed in Method-2 in Section(2.3), weights constructing Method-3 expressed in Section(2.3), a straight line activation expressed in Section(2.4), and the threshold boundaries expressed in Section(2.7). Except the design topology, all the stages of an ANN are expressed. Readers are suggested to read this section.

C.10 The Thresholds and Topology of a Linear Classifier ANN

An Example classifier ANN is developed in Section(2.9) through Section(2.11). The purpose is to demonstrate a concept of design topology for the ANNs that suits with the developments presented in Section(2.8) and expressed in Paragraph(C.9) above.

C.11 Image Data and Extracting Contours

The procedure of finding threshold boundaries of classes of images presented in Section(2.7) and expressed in Paragraph(C.8) is of geometrical nature and should be considered as concept developing. A practicable procedure for the task is presented in Section(2.12). Readers are suggested to read it.

C.12 Finding Limits of Individual Features of a Class of Objects

Unlike the concept of image classification, the objects and events can be detected from their features. And hence the classes of objects/event will be classified from the limits of the individual features. A procedure of finding such limits is presented in Section(2.13).

C.13 From Individual Thresholds to Segment Thresholds

The concept, advantages/options, and a procedure of finding segment thresholds from the individual thresholds is presented in Section(2.14) through Section(2.16). Objects and events may have a large number of components/features. In order to minimise design topology of ANNs, the patterns of features can be divided into suitable segments. There are a number options to select for activation of the segments. This issue is discussed in Section(2.15). A method of finding segment thresholds is given in Section(2.16) with a

software check. However, for a natural example see chapter4 where instead of activating the individual muscle fibbers of a muscle, a group is activated by a single branch of the feeding axon.

C.14 Finding Weights and Time Incremental Learning of Threshold Patterns

The weights construction Method-3 given in Section(2.3) and the concepts developed in Sections(2.4) through Section(2.16), enabled the Sections(2.17-18) to produce meaningful weights and time incremental learning in a simple way.

C.15 Developing a Binary Transform and its Inverse

A binary transform and its inverse is developed in Section(2.19) with a software check. The transform can be effective applied by dividing signals into segments such that the inverse transform can reproduce data with a required accuracy.

C.16 Rejecting Old Methods of Weights Training

It is shown in Section(3.1) that how the author of [1] concluded that there exists no exact method of constructing weights and that the mostly used methods are iterative trial and error methods. And it has already been proved in Section(2.1) that a neuron trained by an error minimisation method is deceive able. If some design topology of a network takes care of not be deceived, even then the methods can not be justified. Because, it is proved in Section(2.2) that the weights successfully produced by the Windrow-Hoff delta rule are nothing more than the scaled versions of the components of the samples. Such a scaling can appropriately be done by exact calculation. For example, the new weights constructing formula given in Equations(2.3.4) provides weights which are the normalised (by energy) versions of samples.

C.17 Hopfield Network do not Qualifies as a Reasonable Network

In Section(3.2), a series of networks has been presented. The Networks are new but may not be recommended for application. The objective of their development is to show that Hopfield network is not a reasonable network. Gradually, it is shown that the weights and activations can not provide anything useful. A fault correction lead to an intermediate

conclusion that the values of weights equals the sample patterns which could simply be stored in memory; hence an ultimate conclusion that the Hopfield network do not qualifies as a reasonable network. Moreover, it is shown in Section(3.3) that the Boltzmann Machine do not qualifies as reasonable network and that its weights after successful training are the shifted versions of the weights of Hopfield network.

C.18 The New Linear Classifier ANN and its Advantages

Although the ANN given in Figure(3.4.1) has been developed with the intention to demonstrate (i) the application of the new weights constructing formula given in Equations(2.3.4), (ii) the linear activation given in Section(2.4), (iii) and the new design topology for classifier ANNs. But blessingly it has been found capable of classifying the scaled versions of the known patterns as well. This ability is due to the straight line activation and dual connections of inputs with the first hidden layer. See as an evidence that the ANN given in Figure(3.5.1) although acts upon the straight lines yet it is unable to tell us about the scaled versions with guarantee.

C.19 A Linear Classifier Acted upon Segment Thresholds

A linear classifier acted upon segment thresholds is presented in Section(3.5) and is shown in Figure(3.5.1). The ANN is developed as an example to demonstrate design topology, activations, and the concepts of Section(2.14) through Section(2.16).

C.20 An XOR ANN and its Activation

A series of XOR ANNs is presented in Section(3.6) and shown in Figure(3.6.1) through Figure(3.6.7). The ANNs are acted upon their threshold decision boundaries. Each of the boundary should neither be consider a high force nor a low force. Because neither it passes fully from above nor fully from below. Instead, a single straight line classifies an event or a class being the only line passing below from all the features of only that event or class. However, for example the upside-down version of Figures(3.6.1), can equivalently be considered as the threshold decision boundary for the classifiers given in Figure(3.6.2). In that case, a single straight line classifies an event or a class being the only line passing above from all the features of only that event or class.

C.21 The Generalised XOR ANN

The XOR ANNs presented in Figure(3.6.6) and Figure(3.6.7) are to detect the happening of one of the n events and to classify one of the n objects when each event or object is able to be isolated by a single straight line. But in real life problems it may not be possible to find sufficient features to be separated in this way. It was therefore required to generalised the XOR logic for the events and the objects that requires more than one straight lines to be isolated from the others. The generalised model is presented in Section(3.8) and shown in Figure(3.8.1).

C.22 The Generalised XOR ANN with Varying Design Topology

The generalisation of the XOR ANN made in Section(3.8) may not be sufficient due to the fact that in real life problems the number of features of events and objects are different. A further generalisation is made in Section(3.9) to fulfil the requirements of the difference in number of features. The ANN is shown in Figure(3.9.2).

C.23 A Neuromuscular ANN Classifier and Segment Thresholds

The Neuromuscular ANN is developed in Section(4.7) and is shown in Figure(4.7). The ANN is a natural example that the classifier ANNs should be acted upon limits of individual components. However, in order to reduce the design topology, ANNs should be acted upon threshold segments. We can modify the activation and the topology of the Neuromuscular ANN to have a reduced classifier ANN. For example, see an ANN shown in Figure(3.5.1).

C24 A Remarkable ANN Classifier

An ANN that can classify objects of m classes has been presented in Figure(5.8). The classifier is able to detect and reconstruct an input pattern as a known or an unknown one. In later case, the pattern is further classified into three categories; a scaled version, or a mirror image, or a noisy version of the sample pattern. Moreover, the ANN is capable to tell us whether the noise is remove able or not. This ANN can be considered as a good example in pattern classification.

C.25 A Computing Redundancy in STFT

A computing redundancy in STFT and a technique of its removal with its software check is presented in Section(6.3). This redundancy is also the issue of wavelet transforms. The non-redundant code can also be found in some other programs presented in Chapters(7.10). Readers are suggested to study Section(6.3) and use non-redundant technique of wavelet placing in all applications.

C.26 Relative Frequency Analysis and Switching from STFT to Wavelet Transform

The drawbacks of STFT listed in paragraphs(2-4) of Section(6.5) were removed by the scientists by linking the STFT with wavelet transform through relative frequency analysis. Readers are suggested to study Section(6.6) for the removal of drawbacks and the connection of STFT with wavelet transform. However, it should be noted that the objective of this thesis is not to deal with the time-frequency analysis.

C.27 The Objective of Wavelet Transform Deduced from STFT

The transform resulted as the consequence of switching from STFT to wavelet transform is of no particular importance towards noise reduction and data compression. Although it does work but it is highly redundant due to the presence of sinusoidal basis of Fourier transform. Therefore, the wavelet transforms presented in Section(6.8) should not be consider useful for noise reduction and data compression. Instead, their purpose is to check the switching and the difference in constant, relative, and constant relative bandwidth families of windows (wavelets).

C.28 Constant Relative Frequency Analysis

The normalised system of Gaussian windows listed in Equation(6.12.2) provides a constant relative bandwidth. Similarly, the new wavelet defined in Equation(7.2.3) provides a constant relative bandwidth family of wavelets. It has been found that a family of constant relative bandwidth wavelets is advantageous in wavelet transforms as compared to constant bandwidth or relative bandwidth or any other system of wavelets. Therefore readers are suggested to study the Section(6.6.1) for the difference of

bandwidths, Section(6.8) for their application, Equation(6.12.2) and Equation(7.2.3) for their mathematical definitions.

C.29 Minimum Number of Gaussian Windows for Wavelet Decomposition

The system of six Gaussian windows presented in Section(6.12) should be consider as a system of appropriate acceptable number of windows to transform signals having length less than or equals 128. Readers are suggested to study sections Software(6.12.4-6) where a software check of this fact is presented with graphical outputs. Processing of larger signals should be done by shifting the system along the signal.

C.30 Status of the Code Presented in Software(6.12.7)

The software program presented in section Software(6.12.7) is neither a simple wavelet decomposition nor a simple wavelet transform. Instead it is a dual wavelet transform associated with the wavelet decomposition; (i) since the system of the windows qualifies as a family of wavelets with relative bandwidths therefore associating a Continuous Wavelet Transform is quite natural, (ii) and since the system of Gaussian windows forms an orthonormal basis therefore associating a Discrete Wavelet Transform is also quite natural. It should be remember that the code do not provides a proper logic of noise reduction or a zoom-in wavelet transform; instead the logic highlights the existence of such transforms.

C.31 Associating two Wavelet Transforms with the Wavelet Decomposition

In Section(6.13), a pair of wavelet transforms has been associated with the wavelet decomposition presented in Section(6.12). The Nature of the wavelet decomposition was found quite suitable for this association. The system of six Gaussian windows has actually been developed to have a family of constant relative bandwidth wavelets for the continuous wavelet transform. But fortunately it was found to be a set of orthonormal basis in bonus with the blessing of God!. The software programs presented in sections Software(6.13.1-3) demonstrate the performance of the DWT and the DIWT while the association of CWT is obvious due to the fact that the system of windows consists of a constant relative bandwidth family of wavelets.

C.32 Difference Between DWT and Wavelet Decomposition

The wavelet decomposition developed in Section(6.12) provides an approximation of a real signal without a guarantee of both the completion and the reconstruction. On the other hand, the DWT developed in Section(6.13) is the same approximation with the guarantee of reconstruction. Readers can observe this difference in both the Figure(6.13.2) and Figure(6.13.3) where the minor information do not appears in DWT/decomposition but appears in DIWT (reconstruction). Moreover, when we use the binary coded version of the Gaussian wavelet packet then DWT differs from the wavelet decomposition. Readers are suggested to note this difference from Figure(6.13.1) and Figure(6.15.2).

C.33 Developing Larger and Denser Wavelet Packets

In Section(6.14), some larger and some denser wavelet packets have been developed and their performance has been checked through software. These packets should be considered as samples to develop larger and denser packets from the given smaller ones.

C.34 The Binary Coded Gaussian Wavelet Packet

The development of Gaussian wavelet packets made in Sections(6.12-15), is the best achievement in Chapter6. Especially, the binary coded wavelet packet being equivalent to six binary numbers is useful to implement it through hardware and is important due to its dual application. It provides orthonormal basis for all functions and at the same time it is a wavelet packet for the continuous wavelet transform. Moreover, it can be used as a single wavelet. Joining the packets from end to end is useful not only to process the lengthy signals but also to manufacture some extend able hardware.

C.35 Binary Coding of Weights is Suggested

It is suggested that the weights of both the high force and the low force thresholding decision boundaries of different classes of objects should be coded to binary form to get advantages of easy processing and implementation of the binary numbers. See for example, the case of Gaussian wavelet packet in Sections(6.15).

C.36 Binary Coding Transform and its Inverse

A binary coding transform with its inverse is presented in Section(3.14). This transform enables us to represent a signal with a single binary number. Readers are suggested to study this section.

C.37 A New Wavelet with a New Defining Style and the Automatic Nature of CWT

The difference of automatic compression and/or expansion between continuous and digital wavelets was found complicated to visualise. If the wavelet compression (expansion) is done by down sampling (up sampling) of the basic wavelet then the width of domain of definition of the wavelet remains the same. This way of defining wavelets is given in Equation(7.2.2), which is advantageous for automatic wavelet compression during zoom-out plus noise reduction but is disadvantageous and misleading for the concept of automatic wavelet expansion during zoom-in wavelet transform. The compression (expansion) of the wavelet should be done by reducing (increasing) the width of the domain of definition which be changed by the resolution parameter j. See for example, a new wavelet defined with this new concept in Equation(7.2.3). This new concept is advantageous for automatic wavelet expansion during zoom-in plus noise reduction wavelet transform but disadvantageous and misleading for automatic wavelet compression during zoom-out plus noise reduction wavelet transform. These issues has been focused in detail in Section(7.2). Readers are strongly suggested to study this section.

C.38 A Noise Reducing Stand Alone ANN

Although noise reduction is done automatically during both the zoom-in and the zoom-out wavelet transforms. However, in some applications we require only noise reduction. That is, we require a stand alone noise reducing ANN. There are two slightly different CWT presented in Section(7.3) and Section(8.2). Readers are suggested to note the difference that the transform presented in Section(8.2) is non-redundant in the sense expressed in Section(6.3). An ANN for the non-redundant transform is shown in Figure(8.2.1) and it software check is shown in Figure(8.2.2).

C.39 Automatic Wavelet Compression and Data Expansion (multiresolution zoom-out DWT)

Wavelets should automatically be compressed during a zoom-out plus noise reduction wavelet transform. Readers should be careful that there are two different styles of defining wavelets which are expressed in Section(7.2) in detail. Unlike the style given in Relation(7.2.3), the style given in Relation(7.2.2) and Relation(7.2.4) suits for zoom-out plus noise reducing transform. A zoom-out plus noise reducing wavelet transform and its inverse transform has been presented in Section(7.4). Moreover, "Software and Description of Automatic Nature of Zoom-out CWT" is presented as a subsection with its graphical outputs. Readers are strongly suggested to study this subsection in order to understand the behaviour of zoom-out plus noise reduction wavelet transform and the automatic wavelet compression during the transform. The software is non-redundant in the sense described in Section(6.3).

C.40 Automatic Wavelet Expansion and Data Compression (multiresolution zoom-in DWT)

Wavelets should automatically be expanded during a zoom-in plus noise reduction wavelet transform. Readers should be careful that there are two different styles of defining wavelets and are expressed in Section(7.2) in detail. Unlike the style given in Relation(7.2.2) and Relation(7.2.4), the style given in Relation(7.2.3) suits for zoom-in plus noise reducing transform. A zoom-in plus noise reducing wavelet transform and its inverse transform has been presented in Section(7.5). Moreover, a subsection Software(7.5) is presented with graphical support shown in Figure(7.5.1). Readers are suggested to study this section in order to understand the behaviour of zoom-in plus noise reduction wavelet transform and the automatic wavelet compression during the transform. The software is non-redundant in the sense described in Section(6.3).

C.41 Two ways of defining Zoom-in and Zoom-out Wavelet Transforms and a Confusion

By ignoring the noise reducing stand alone wavelet transform presented in Section(7.3) and Section(8.2), the CWT can be divided into two transforms; a zoom-in plus noise reducing and a zoom-out plus noise reducing wavelet transform. Each of the two transforms can either be defined directly from the Relation(7.1.4) and/or Relation(8.1.2)

or otherwise by inverting the one to have the other. Readers are suggested to see both of these ways in Section(7.4) and Section(7.5) and should be careful for the concept of inversion. The inverse transform of the one results in defining the other which does not means reversion or reconstruction.

C.42 A CWT is irreversible unlike a DWT

Unlike the Discrete Wavelet Transform presented in Section(6.13), a Continuous Wavelet transform can not be reversed to produce the input data. It is therefore necessary to be careful about the inversion of both the zoom-out plus noise reduction and the zoom-in plus noise reduction wavelet transforms. However, the New zoom-out wavelet transforms presented in Section(8.6), Section(10.1.2), and Section(10.1.4) are reversible.

C.43 Multi-Option ANNs are Suggested

In practical problems we need to process signals and images for classification or/and noise reduction or/and data compression. It is found advantageous to design multi-option and multi-purpose automatic ANNs. See for example the ANN given in Figure(7.6.2).

C.44 A Modified Zoom-out plus Noise Reducing ANN

The zoom-out plus noise reducing wavelet transform presented in Section(7.4) has been modified by shifting the resolution and translation parameters from wavelet to the signal so that the size of the wavelet can not be changed during the transform. This modification required non-available values to be inserted in between each pair of the components of the signal. A timely fix to this problem was found in dropping the factor $1/2^j$ from $f(i+k/2^j)$ to have $f(i+k)$ and then repeating 2^j times the weighted sum of each window placing to have a magnification of order 2^j. Readers are suggested to observe its performance from the software **tnrzo.cpp** whose outputs have been shown in Figure(8.1.1).

C.45 A Modified Zoom-in plus Noise Reducing ANN

The zoom-in plus noise reducing wavelet transform presented in Section(7.5) has been modified by shifting the resolution and translation parameters from wavelet to the signal

so that the size of the wavelet can not be changed during the transform. The ANN of this modified transform has been shown in Figure(8.3.1) and its performance is shown in Figure(8.3.2). Readers are suggested to find this transform better than that presented in Section(7.5).

C.46 A New Zoom-out Wavelet Transform

Recall that the new zoom-out transform has dual application; one is the data expansion described in Section(8.6) and other is the data reconstruction between adjacent peaks and bottoms which is described in Section(10.1.2) and Section(10.1.4). For both of these purposes a new zoom-out wavelet transform is presented in Section(8.6).

C.47 A Wavelet Noise Reducing and A Wavelet Data Compressing Feedback ANNs

Two switch-able feedback ANNs have been presented in Section(8.7) and Section(8.8) and are shown in Figure(8.7.1) and Figure(8.8.1) respectively. The purpose is to repeat the processing so that a required level of noise reduction and data compression be achieved. This type of ANNs can be useful to process signals having a diverse signal/noise ratio and frequency contents.

C.48 Noise Reducing Plus Data Compressing ANNs and a New Wavelet

A conclusion that " the repeated application of the average smoothing filter is equivalent to a wavelet transform" has been made in Section(9.1). Therefore the new wavelet defined in Section(7.2) is justified and should be used in ANNs as given in Figures(8.2.1)-(8.4.1).

C.49 Two-dimensional Data Smoothing Wavelet Transform and its ANNs

A two-dimensional data smoothing wavelet transform is presented with software support in Section(9.3). Its Mathematical formula remains to be established. However, its ANN has shown in Figure(9.3.1) which is a sample. On the same style some higher order ANNs should be designed for data smoothing.

C.50 Two-dimensional Data Compressing Wavelet Transform and its ANNs

A two-dimensional data compressing wavelet transform having two options has been presented in Section(9.6) with software support. Its ANN should be design on the style of the ANN shown in Figure(9.3.1) but with step-size as require. The one-dimensional data compressing wavelet transform accompanied with their ANNs have already been presented in Chapters(7-8) which are samples for the two-dimensional ANNs as well.

C.51 Modifying the New Zoom-out Wavelet Transform

A modification in the new zoom-out wavelet transform has been made in Section(10.2) as a consequence of the observation of its performance for reconstruction of data between peaks and bottoms of a cosine signal shown in Figure(10.2). This modification is interesting and readers are suggested to study it.

C.52 Peaks and Bottom Detecting ANN

Two ANNs for detection of peaks of cusps and bottoms of troughs of signals have been presented in Section(10.6-8) and Section(10.11). It is suggested that the ANN presented in Section(10.6-8) should be considered just for concept developing, because in Section(10.9) it is shown with software check that the segment-length equals 8 may not work well for high frequency contents. Moreover, in the same Section(10.9) and Section(10.10) it is shown that the segment-length equals 4 is the best. It is therefore suggested that the ANN presented in Section(10.11) be considered a final version.

C.53 Dual Circuit for Peaks and Bottoms Detecting ANN

It is found that the logic of detecting peaks and bottoms differs only for the activation of first hidden layer of the circuit-segments shown in both the Figure(10.8) and Figure(10.13). It is therefore suggested that the circuit-segments be designed of dual nature as shown in Figure(10.14).

C.54 Classification and Prominent Features

Although classification can be done with number of peaks and bottoms. But when the full accuracy and reconstructed is required, then picking amplitudes and indices of peaks and

bottoms is sufficient for both the classification and reconstruction. It is also sufficient even when the features of patterns are variant as described in Section(5.2). A number of ANNs with slightly different design topology and activation can be developed utilising these features. As a sample, a basic structure of such an ANN is shown in Figure(5.4) and a refined model is shown in Figure(5.7). A particular example of the model is presented in Figure(5.8) with a software check. However, it is advantageous to point out the logic of

(i) an ANN having the capability of measuring the differences of indices of all couples of adjacent peaks, or bottoms, or (peak, bottom) of an input pattern will be able to classify it as a compressed/expanded version of the known pattern. If it finds that all the differences lie in a small specified range and then if input/sample ratio of the average difference is grater (smaller) than one, the pattern will be an expanded (compressed) version of the known pattern. Such an ANN is presented in Section(10.16) and its model is shown in Figure(10.16).

(ii) Likewise, if it finds that all the differences of amplitudes lie in the specified range and then if the input/sample ratio of the average difference is grater (smaller) than one, the pattern will be a brighter (darker) version of the known image or an scaled up (down) version of the known signal. An example software code can be seen in **tremann.cpp** along with the design topology of the ANN in Section(5.8).

C.55 Extracting Prominent Contours from Image Data is Suggested

Recall the developments made in Chapter5 and Chapter10 where the issues of detecting prominent features of signals and images are presented. Especially, detection of peaks and bottoms and picking their amplitudes and indices is important for classification and data reduction with the guarantee of reconstruction. In addition to this, for images we can have further data reduction and can enhance the classification. It has been observed practically that both the peaks and the bottoms of rows of an image have some meaningful continuation from row to row. See for example, the output data by executing the software program **tfpidz.cpp** listed in Appendix(5.6). This continuation forms contours. Mathematically, a contour is a piece-wise continuous curve represented by a complex-valued function of a real variable. These contours should be divided (if required)

into sections (the high force and the low force thresholding boundaries) such that no two or more than two peaks or bottoms lie on a vertical straight line when drawn on a section. A number of advantages can be achieved: (i) the sections can further be processed for data reduction by picking their corner points and discarding the data between each pair of corners, (ii) picking their relative positions will result in getting ride of the difference of physical positions of samples and the actual images at the scenes as a consequence the task of classification will be simplified.

C.56 Classification and Number of Waves

Classification can possibly be done by detecting number of waves in a signal (and in each row of an image) plus a two bit binary number showing the occurrence of a peak/bottom at the start and at the end of the signal. The procedure of this task has been presented with a software check in section Software(10.1.2). It is suggested to design an ANN to perform this task as shown Figure(10.13).

C.57 Bit by bit Adders and Multiplexors

It is mentioned in Chapter11 that millions of processors are required for both the task of classification and noise reduction plus data compression through the ANNs to be designed. We need to develop small parallel adders and multiplexors. In this connection there are two type of parallelism. The usual type which is adopted in wavelet transforms and their ANNs, and the unusual type which is in addition to the usual one. The unusual type expresses that instead of applying the usual addition and multiplication operators, which are no doubt sequential being the binary ones, each weighted sum should be computed through bit by bit addition. This type of parallelism has been explained in Chapter11. A number of circuit-units have been presented to built larger circuits. As an example a larger circuit is presented in Figure(11.9). Any required circuit can be developed by assembling the small units in the same way. Moreover, to process floating points such units and the larger circuits are suggested to develop.

References

1. R Beale, T Jackson, *Neural Computing*, BP 15-163, 1994.
2. Timothy Masters, USA, *Signal and Image Processing with Neural Networks*, 1994, BP 1-76.
3. Timothy Masters, USA, *Practical Neural Networks Recipes in C++*, BP 1-276, 1993.
4. LiMin Fu, *Neural Networks in Computer Intelligence*, BP 18-110, 1994.
5. John V. Basmajian, Carlu J. De Luca; *Muscle alive(Their functions revealed by Electromyography)*; BP 1-180;
6. Charles K. Chui, Department of Mathematics, Texas A&M University, College Station, Texas; *An Introduction to Wavelets*, 1995.
7. Adhemar Bultheel, *Leaning to Swim in a Sea of Wavelets*, Bull. Belg. Math. Soc. 2(1995), PP1-44.
8. Olivier Rioul and Martin Vetterli, *Wavelets and Signal Processing,* IEEE SP Magazine, October 1991, PP14-38.
9. Alain Fournier and Leena-Maija Reissell, University of British Columbia, *Wavelets and their Applications in Computer Graphics,* Lecture notes given at Siggraph '94 Conference, 1994, PP1-44.

Appendix 1

Appendix 1.1

```
// tfft1d.cpp
// This program is developed to observe the results of Foureir transform.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>

#include "nrzozi.h"

#define sigsize 32

void main(void)
{
        int i,Xpos,Xstep,Ystep;
        float *s,*si;

        s=(float *)calloc(sigsize+1, sizeof(float));
        si=(float *)calloc(sigsize+1, sizeof(float));

        Xstep=2;
        Ystep=-20;

        INITIAL();
        SCRBLACK();

        COSINE(s,sigsize,4);XPLOT(s,sigsize,Xstep,Ystep,20,80,GREEN);
        FFT1D(s,si,sigsize,1);XPLOT(s,sigsize,Xstep,5*Ystep,20,160,CYAN);
        FFT1D(s,si,sigsize,-1);XPLOT(s,sigsize,Xstep,Ystep,20,260,LIGHTGRAY);

        TRIANGLE(s,sigsize,16);NXPLOT(s,sigsize,Xstep,Ystep,120,80,GREEN);
        FFT1D(s,si,sigsize,1);NXPLOT(s,sigsize,Xstep,5*Ystep,120,160,CYAN);
        FFT1D(s,si,sigsize,-1);NXPLOT(s,sigsize,Xstep,Ystep,120,260,LIGHTGRAY);

        TOPHAT(s,sigsize,8);NXPLOT(s,sigsize,Xstep,Ystep,220,80,GREEN);
        FFT1D(s,si,sigsize,1);NXPLOT(s,sigsize,Xstep,5*Ystep,220,160,CYAN);
        FFT1D(s,si,sigsize,-1);NXPLOT(s,sigsize,Xstep,Ystep,220,260,LIGHTGRAY);

        SPIKES(s,sigsize,8);NXPLOT(s,sigsize,Xstep,Ystep,320,80,GREEN);
        FFT1D(s,si,sigsize,1);NXPLOT(s,sigsize,Xstep,5*Ystep,320,160,CYAN);
        FFT1D(s,si,sigsize,-1);NXPLOT(s,sigsize,Xstep,Ystep,320,260,LIGHTGRAY);

        getch();
        free (s);free (si);
        closegraph();

}
```

Appendix 1.2

```
// tstftp.cpp
// To demonstrate (i) the procedure of sliding a window along a signal,
//                (ii) the spread of frequency contents over the range,
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include "nrzozi.h"

#define n 128
#define phi0 .01
#define t0 1

void main(void)
{
      int i,T,p,q=2;
      int Xstep,Ystep,wstep,width;
      float *f,*g,*sr,*si;

      f=(float *)calloc(n+2, sizeof(float));
      g=(float *)calloc(n+2, sizeof(float));
      sr=(float *)calloc(n+2, sizeof(float));
      si=(float *)calloc(n+2, sizeof(float));

      Xstep=2;
      Ystep=-20;
      INITIAL();
      SCRBLACK();

      SINE(f,n,8);XPLOT(f,n,Xstep,Ystep,1,40,GREEN);
      wstep=8;
      width=4;
      for(p=0;p<n-width-wstep; p=p+wstep){
            T=p*t0;
            GAUSSIAN(g,n,width,T);XPLOT(g,n,Xstep,10*Ystep,1,120,YELLOW);
            for(i=1;i<=n;i++){sr[i]=g[i]*f[i];si[i]=0;}
            XPLOT(sr,n,Xstep,10*Ystep,1,170,GREEN);
            STFT(sr,si,n,1,q,phi0);
            for(i=1;i<=n;i++)si[i]=0;
            STFT(sr,si,n,-1,q,phi0);XPLOT(sr,n,Xstep,10*Ystep,1,340,LIGHTGRAY);
      }
      // Compute and display the graph of the slice of windowed sine
      GAUSSIAN(g,n,width,T);
      for(i=1;i<=n;i++){sr[i]=g[i]*f[i];si[i]=0;}
      STFT(sr,si,n,1,q,phi0);XPLOT(sr,n,Xstep,300*Ystep,1,250,CYAN);

      getch();
      free (sr); free (si); free (f); free (g);
      closegraph();
}
```

Appendix 1.3

```
// tstftq.cpp
// To demonstrate that the suitable values of
// frequency parameter q ranges from -0.04 to +0.04.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include "nrzozi.h"
#define n 128
#define phi0 .01

void main(void)
{
        int i,T,p=1,q;
        int Xstep,Ystep,wstep,width,t0=1;
        float *f,*g,*sr,*si;

        f=(float *)calloc(n+2, sizeof(float));
        g=(float *)calloc(n+2, sizeof(float));
        sr=(float *)calloc(n+2, sizeof(float));
        si=(float *)calloc(n+2, sizeof(float));

        Xstep=2; Ystep=-16;
        INITIAL();
        SCRBLACK();

        SINE(f,n,16);XPLOT(f,n,Xstep,Ystep,1,30,GREEN);
        width=16; T=32*p*t0;
        GAUSSIAN(g,n,width,T);XPLOT(g,n,Xstep,30*Ystep,1,30,YELLOW);
        for(i=1;i<=n;i++){sr[i]=g[i]*f[i];si[i]=0;}
        XPLOT(sr,n,Xstep,30*Ystep,1,75,GREEN);

        for(q=-20;q<=20;q=q+2){
              for(i=1;i<=n;i++){sr[i]=g[i]*f[i];si[i]=0;}
              STFT(sr,si,n,1,q,phi0);XPLOT(sr,n,Xstep,500*Ystep,1,150,CYAN);
                                XPLOT(si,n,Xstep,500*Ystep,1,200,CYAN);
              // The case of ISTFT for which si[] is not set equals zero.
              STFT(sr,si,n,-1,q,phi0);XPLOT(sr,n,Xstep,50*Ystep,1,270,LIGHTGRAY);
              for(i=1;i<=n;i++){sr[i]=g[i]*f[i];si[i]=0;}
              STFT(sr,si,n,1,q,phi0);XPLOT(sr,n,Xstep,500*Ystep,1,350,CYAN);
              for(i=1;i<=n;i++)si[i]=0;

              // The case of ISTFT when si[] is set equals zero.
              STFT(sr,si,n,-1,q,phi0);XPLOT(sr,n,Xstep,50*Ystep,1,425,LIGHTGRAY);

              getch();setviewport( 0, 110, 638, 450, 1);clearviewport();
        }
        free (sr); free (si); free (f); free (g);
        closegraph();
}
```

Appendix 1.4

```
// tstftpf.cpp
// To demonstrate (i) the fixation of the spread of frequncey contents,
//                (ii) the removal of computing redundancy from STFT.
// Author: M. D. Choudhry
#include <graphics.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>
#include "nrzozi.h"
#define n 128

void main(void)
{
      int i,j,k,p,np,nn,r,a,dw, width,Xstep,Ystep,Xpos,tstep;
      char ch;
      float *f,*g,*gh,*s,*si;

      f=(float *)calloc(n+1, sizeof(float));
      Xstep=2;Ystep=-30;
      INITIAL();
      for(r=5;pow(2,r)<=n;r++){
            width=pow(2,r);
            g=(float *)calloc(width+1, sizeof(float));
            gh=(float *)calloc(width+1, sizeof(float));
            s=(float *)calloc(width+1, sizeof(float));
            si=(float *)calloc(width+1, sizeof(float));
            SCRBLACK();
            COSINE(f,n,8);XPLOT(f,n,Xstep,Ystep,1,75,GREEN);
            MGAUSS(g,width);
            tstep=width/4;
            nn=(n-width)/tstep;
            for(p=0;p<=nn;p++){
                  Xpos=Xstep*tstep*p;
                  XPLOT(g,width,Xstep,Ystep,Xpos,75,YELLOW);
                  for(i=1,k=1;i<=width;k=k+1,i++){
                        s[k]=g[i]*f[i+tstep*p]; si[k]=0.0;
                        gh[k]=g[i];
                  }
                  dw=width;
                  Xpos=Xstep*(tstep)*p;
                  XPLOT(gh,dw,Xstep,Ystep,Xpos,175,YELLOW);
                  XPLOT(s,dw,Xstep,Ystep,Xpos,175,GREEN);
                  FFT1D(s,si,dw,1); XPLOT(s,dw,Xstep,3*Ystep,Xpos,260,CYAN);
                  FFT1D(s,si,dw,-1); XPLOT(s,dw,Xstep,Ystep,Xpos,350,LIGHTGRAY);
                  getch();
            }
            free (s);free (si); free (g);free (gh);
      }
      closegraph();
      free (f);
}
```

Appendix 1.5

```
// trband.cpp
// To demonstrate the formatiom of windows having
// (i)  constant bandwidths,
// (ii) relative bandwidths,
// (iii) constant relative bandwidths.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "nrzozi.h"
#define n 128

void main(void)
{
      int j,p,nn,a;
      int width,Xstep,Ystep,Xpos,tstep;
      float *g1,*g2;

      g2=(float *)calloc(2*n+1, sizeof(float));

      Xstep=2;
      Ystep=-20;
      INITIAL();
      SCRBLACK();

      for(j=5;pow(2,j)<=n;j++){
            width=pow(2,j);
            g1=(float *)calloc(width+1, sizeof(float));
            MGAUSS(g1,width);
            tstep=width/4;
            nn=(n-width)/tstep;
            for(p=0;p<=nn;p++){
                  Xpos=Xstep*tstep*p;
                  XPLOT(g1,width,Xstep,Ystep,Xpos,(j-4)*70,GREEN);
            }
      }
      for(j=1;pow(2,j)<=n/4;j++){
            width=pow(2,j);
            GAUSS11(g2,n,width,2*j);
            XPLOT(g2,n,Xstep,16*Ystep,1,330,GREEN);
            XPLOT(g2,n,Xstep,2*width*Ystep,1,400,GREEN);
      }
      getch();
      closegraph();
      free (g1);
      free (g2);
}
```

Appendix 1.6

```
// trbwt11.cpp
// To demonstrate relative bandwidth wavelet transform.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,Xpos,Xstep,Ystep;
        float *a,*g,*sr,*si,*fr,*fi,width;

        a=(float *)calloc(n+1, sizeof(float));
        g=(float *)calloc(n+1, sizeof(float));
        sr=(float *)calloc(n+1, sizeof(float));
        si=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(n+1, sizeof(float));
        fi=(float *)calloc(n+1, sizeof(float));

        Xpos=1; Xstep=2; Ystep=-20;

        INITIAL();
        SCRBLACK();
        COSINE(a,n,16);XPLOT(a,n,Xstep,Ystep,Xpos,100,GREEN);

        for(j=1;pow(2,j)<=n/2;j++){
                width=pow(2,j);
                GAUSSCR(g,n,width,0);
                XPLOT(g,n,Xstep,16*Ystep,1,100,YELLOW);
                for(i=1;i<=n;i++){
                        sr[i]=20*g[i]*a[i];
                        si[i]=0;
                }
                XPLOT(sr,n,Xstep,Ystep/2,Xpos,150,GREEN);
                FFT1D(sr,si,n,1);
                XPLOT(sr,n,Xstep,6*Ystep,Xpos,j*30+200,CYAN);
                for(i=1;i<=n;i++){
                        fr[i]=fr[i]+width*sr[i];
                }
                getch();
        }
        FFT1D(fr,fi,n,-1);
        XPLOT(fr,n,Xstep,Ystep/10,Xpos,420,LIGHTGRAY);
        getch();
        free (a); free (g);free (sr); free (si);free (fr); free (fi);
        closegraph();
}
```

Appendix 1.7

```
// tcrbwt11.cpp
// To demonstrate constant relative bandwidth wavelet transform.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,Xpos,Xstep,Ystep;
        float *a,*g,*sr,*si,*fr,*fi,width;

        a=(float *)calloc(n+1, sizeof(float));
        g=(float *)calloc(n+1, sizeof(float));
        sr=(float *)calloc(n+1, sizeof(float));
        si=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(n+1, sizeof(float));
        fi=(float *)calloc(n+1, sizeof(float));

        Xpos=1; Xstep=2; Ystep=-20;
        INITIAL();SCRBLACK();

        COSINE(a,n,16);XPLOT(a,n,Xstep,Ystep,Xpos,50,GREEN);
        for(j=1;pow(2,j)<=n/2;j++){
                width=pow(2,j);
                GAUSSCR(g,n,width,0);
                XPLOT(g,n,Xstep,2*width*Ystep,1,50,YELLOW);
                for(i=1;i<=n;i++){
                        sr[i]=2*width*g[i]*a[i];
                        si[i]=0;
                }
                XPLOT(sr,n,Xstep,Ystep,Xpos,100,GREEN);
                FFT1D(sr,si,n,1);
                XPLOT(sr,n,Xstep,10*Ystep,Xpos,j*50+85,CYAN);
                for(i=1;i<=n;i++){
                        fr[i]=fr[i]+sr[i];
                }
                getch();
        }
        FFT1D(fr,fi,n,-1);
        XPLOT(fr,n,Xstep,Ystep,Xpos,430,LIGHTGRAY);
        getch();

        free (a); free (g);free (sr); free (si);free (fr); free (fi);
        closegraph();
}
```

Appendix 1.8

```
// tmrwt11.cpp
// To demonstrate multiresolution wavelet transform with relative bandwidth.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>
#include "nrzozi.h"
#define n 128

void main(void)
{
      int i,j,k,r,a,dw, width,Xstep,Ystep,Xpos;
      float *f,*g,*sr,*si,*fr,*fi;

      f=(float *)calloc(n+1, sizeof(float));
      g=(float *)calloc(n+1, sizeof(float));
      sr=(float *)calloc(n+1, sizeof(float));
      si=(float *)calloc(n+1, sizeof(float));
      fr=(float *)calloc(n+1, sizeof(float));
      fi=(float *)calloc(n+1, sizeof(float));

      Xpos=1; Xstep=2; Ystep=-30;
      INITIAL();SCRBLACK();

      for(j=0;j<=2;j++){
            a=pow(2,j);
            COSINE(f,n,8); XPLOT(f,n,Xstep,Ystep,1,180,GREEN);
            for(r=2;pow(2,r)<=n;r++){
                  width=pow(2,r);
                  GAUSSCR(g,n,width/2,0); XPLOT(g,n,Xstep,16*Ystep,Xpos,180,YELLOW);
                  for(i=1,k=1;i<=n;k=k+1,i=i+a){
                      sr[k]=g[i]*f[i];
                   }
                   dw=n/a;
                   XPLOT(sr,dw,Xstep,10*Ystep,Xpos,250,GREEN);
                   FFT1D(sr,si,dw,1);
                   for(i=1;i<=dw;i++){
                      fr[i]=fr[i]+width*sr[i];
                   }
            }
            FFT1D(fr,fi,dw,-1); XPLOT(fr,dw,Xstep,Ystep/2,Xpos,350,LIGHTGRAY);
            for(i=1;i<=dw;i++){fr[i]=0;fi[i]=0;}
            getch();
            SCRBLACK();
      }
      closegraph();
      free (f);free (g);free (sr);free (si);free (fr);free (fi);
}
```

Appendix 1.9

```
// tw7gauss.cpp
// To Demonstrate the behaviour of 7th
// constant relative window of gaussian.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

#include "nrzozi.h"
#define n 128

void main(void)
{
        int Xpos,Xstep,Ystep;

        float *g,width;

        g=(float *)calloc(n+1, sizeof(float));

        Xpos=1;
        Xstep=2;
        Ystep=-4;

        INITIAL();
        SCRBLACK();

        width=pow(2,7);
        GAUSSCR(g,n,width,7);
        XPLOT(g,n,Xstep,2*width*Ystep,Xpos,150,MAGENTA);

        getch();

        free (g);
        closegraph();

}
```

Appendix 1.10

```
// twdcband.cpp
// To demonstrate the constant bandwidth wavelet decomposition of a signal (cosine).
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>
#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,k,p,nn,r,dw;
        int width,Xstep,Ystep,Xpos,tstep;
        float *f,*fr,*g,*s;

        f=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(n+1, sizeof(float));
        g=(float *)calloc(n+1, sizeof(float));
        s=(float *)calloc(n+1, sizeof(float));

        Xstep=2; Ystep=-16;
        INITIAL();SCRBLACK();
        COSINE(f,n,8);
        for(r=4,j=0;pow(2,r)<=n/2;j=j+140,r++){
                width=pow(2,r);
                XPLOT(f,n,Xstep,Ystep,1,40+j,GREEN);
                MGAUSS(g,width);
                tstep=width/4;
                nn=(n-width)/tstep;
                for(p=0;p<=nn;p++){
                        Xpos=Xstep*tstep*p;
                        XPLOT(g,width,Xstep,Ystep,Xpos,40+j,MAGENTA);
                        for(i=1,k=1;i<=width;k=k+1,i++){
                                s[k]=g[i]*f[i+tstep*p];
                                fr[i+tstep*p]=fr[i+tstep*p]+s[k];
                        }
                        dw=width;
                        Xpos=Xstep*(tstep)*p;
                        XPLOT(s,dw,Xstep,Ystep,Xpos,85+j,CYAN);
                }
                XPLOT(fr,n,Xstep,Ystep,1,120+j,LIGHTGRAY);
                for(i=1;i<=n;i++)fr[i]=0.;
        }
        getch();
        closegraph();
        free (fr);free (f);free (s);free (g);
}
```

Appendix 1.11

```
// twdcomp1.cpp
// To demonstrate wavelet decomposition by using cosine signal
// and constant relative bandwidth Gaussian windows.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

#include "nrzozi.h"
#define n 64

void main(void)
{
        int i,j,Xpos,Xstep,Ystep,width;
        float *f,*g,*sr,*fr;

        f=(float *)calloc(n+1, sizeof(float));
        g=(float *)calloc(n+1, sizeof(float));
        sr=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(n+1, sizeof(float));

        Xpos=1; Xstep=4;
        Xstep=4; Ystep=-20;

        INITIAL();
        SCRBLACK();
        COSINE(f,n,8);XPLOT(f,n,Xstep,Ystep,Xpos,70,GREEN);
        for(j=1;j<=10;j++){
              width=2*j;
              GAUSSCR(g,n,width,j);
              XPLOT(g,n,Xstep,2*width*Ystep,1,70,YELLOW);
              for(i=1;i<=n;i++){
                    sr[i]=width*g[i]*f[i];
                    fr[i]=fr[i]+sr[i];
              }
        }
        XPLOT(fr,n,Xstep,Ystep,Xpos,165,LIGHTGRAY);
        for(i=1;i<=n;i++){
              sr[i]=f[i]-fr[i];
        }
        XPLOT(sr,n,Xstep,Ystep,Xpos,270,MAGENTA);
        for(i=1;i<=n;i++){
              f[i]=fr[i]+sr[i];
        }
        XPLOT(f,n,Xstep,Ystep,Xpos,350,LIGHTGRAY);
        getch();
        free (f); free (sr); free (g);free (fr);
        closegraph();
}
```

Appendix 1.12

```
// twdcomp2.cpp
// To demonstrate wavelet decomposition by using cosine
// signal and constant relative bandwidth Gaussian windows.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

#include "nrzozi.h"
#define n 64

void main(void)
{
        int i,j,Xpos,Xstep,Ystep,width;
        float *a,*g,*sr,*fr;

        a=(float *)calloc(n+1, sizeof(float));
        g=(float *)calloc(n+1, sizeof(float));
        sr=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(n+1, sizeof(float));

        Xpos=1; Xstep=4; Ystep=-30;

        INITIAL();
        SCRBLACK();
        COSINE(a,n,8);XPLOT(a,n,Xstep,Ystep,Xpos,70,GREEN);
        for(j=1;j<=6;j++){
              width=pow(2,j);
              GAUSSCR(g,n,width,j);
              XPLOT(g,n,Xstep,2*width*Ystep,1,70,YELLOW);
              for(i=1;i<=n;i++){
                    sr[i]=width*g[i]*a[i];
                    fr[i]=fr[i]+sr[i];
              }
        }
        XPLOT(fr,n,Xstep,Ystep,Xpos,150,LIGHTGRAY);
        for(i=1;i<=n;i++){
              sr[i]=a[i]-fr[i];
        }
        XPLOT(sr,n,Xstep,Ystep,Xpos,230,5);
        for(i=1;i<=n;i++){
              a[i]=fr[i]+sr[i];
        }
        XPLOT(a,n,Xstep,Ystep,Xpos,320,LIGHTGRAY);

        getch();
        free (a); free (g);free (sr); free (fr);
        closegraph();
}
```

Appendix 1.13

```
// twdcomp3.cpp
// To demonstrate wavelet decomposition by using cosine
// signalby and constant relative bandwidth Gaussian windows.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,Xpos,Xstep,Ystep,width;
        float *a,*g,*sr,*fr;

        a=(float *)calloc(n+1, sizeof(float));
        g=(float *)calloc(2*n+1, sizeof(float));
        sr=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(n+1, sizeof(float));

        Xpos=1; Xstep=2; Ystep=-30;

        INITIAL();
        SCRBLACK();
        COSINE(a,n,8);XPLOT(a,n,Xstep,Ystep,Xpos,70,GREEN);
        for(j=1;j<=6;j++){
              width=pow(2,j);
              GAUSSCR(g,n,width,j);
              XPLOT(g,n,Xstep,2*width*Ystep,1,70,YELLOW);
              for(i=1;i<=n;i++){
                    sr[i]=width*g[i]*a[i];
                    fr[i]=fr[i]+sr[i];
              }
        }
        XPLOT(fr,n,Xstep,Ystep,Xpos,150,LIGHTGRAY);
        for(i=1;i<=n;i++){
              sr[i]=a[i]-fr[i];
        }
        XPLOT(sr,n,Xstep,Ystep,Xpos,230,5);
        for(i=1;i<=n;i++){
              a[i]=fr[i]+sr[i];
        }
        XPLOT(a,n,Xstep,Ystep,Xpos,320,LIGHTGRAY);

        getch();
        free (a); free (sr); free (fr); free (g);
        closegraph();
}
```

Appendix 1.14

```
// twdcomp4.cpp
// To demonstrate relative bandwidth wavelet decomposition
// by using cosine signal and Gaussian window.
// Author: M. D. Choudhry
#include <graphics.h>
#include <stdlib.h>
//#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,hn,Xpos,Xstep,Ystep,width;
        float *g,*a,*fr,*sr;

        a=(float *)calloc(n+1, sizeof(float));
        g=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(n+1, sizeof(float));
        sr=(float *)calloc(n+1, sizeof(float));
        Xpos=1; Xstep=2; Ystep=-30;
         INITIAL();SCRBLACK();
        COSINE(a,n,8);XPLOT(a,n,Xstep,Ystep,Xpos,70,GREEN);
        for(j=1;j<=6;j++){
              width=pow(2,j);
              GAUSSCR(g,n,width,j);
              XPLOT(g,n,Xstep,2*width*Ystep,Xpos,70,YELLOW);
              for(i=1;i<=n;i++){
                    sr[i]=width*g[i]*a[i];
                    fr[i]=fr[i]+sr[i];
              }
        }
        hn=n/2;
        for(i=1;i<=hn;i++){
             sr[i]=width*g[i]*a[i];
             fr[i+hn]=fr[i+hn]+sr[i];
        }
        XPLOT(g,hn,Xstep,2*width*Ystep,Xstep*hn,70,YELLOW);
        XPLOT(fr,n,Xstep,Ystep,Xpos,150,LIGHTGRAY);

        for(i=1;i<=n;i++){
              sr[i]=a[i]-fr[i];
        }
        XPLOT(sr,n,Xstep,Ystep,Xpos,230,5);
        for(i=1;i<=n;i++){ a[i]=fr[i]+sr[i]; }
        XPLOT(a,n,Xstep,Ystep,Xpos,320,LIGHTGRAY);
        getch();
        free (a);free (g);free (fr);free (sr);
        closegraph();
}
```

Appendix 1.15

```
// twdmrrb.cpp
// To demonstrate multiresolution wavelet decomposition
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "nrzozi.h"

#define n 128

void main(void)
{
        int i,j,a;
        int width,Xstep,Ystep,Xpos;
        float *f,*fr;

        f=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(n+1, sizeof(float));

        Xpos=1;
        Xstep=2;
        Ystep=-10;

        INITIAL();
        SCRBLACK();

        TOPHAT(f,n,32);
        XPLOT(f,n,Xstep,Ystep,1,60,GREEN);

        for(j=1;j<=3;j++){
                a=pow(2,j);
                for(i=0;i<=n;i++)fr[i]=0;
                WDMRRB(f,fr,n,j);
                XPLOT(fr,n/a,Xstep,Ystep,Xpos,j*40+60,CYAN);
        }

        getch();

        closegraph();

        free (f);
        free (fr);

}
```

Appendix 1.16

```
// twt&wit1.cpp
// To demonstrate wavelet transform and its inverse
// by using constant relative bandwidth Gaussian windows.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>

#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,Xpos,Xstep,Ystep,width;
        float *a,*g,*sr,*fr,*f;

        a=(float *)calloc(n+1, sizeof(float));
        g=(float *)calloc(n+1, sizeof(float));
        sr=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(n+1, sizeof(float));
        f=(float *)calloc(n+1, sizeof(float));

        Xpos=1; Xstep=2; Ystep=-16;

        INITIAL();
        SCRBLACK();

        COSINE(a,n,8);
       XPLOT(a,n,Xstep,Ystep,Xpos,70,GREEN);

        for(j=1;j<=6;j++){
              width=pow(2,j);
              GAUSSCR(g,n,width,j);
              XPLOT(g,n,Xstep,2*width*Ystep,1,70,MAGENTA);
              for(i=1;i<=n;i++){
                    sr[i]=width*g[i]*a[i];
                    fr[i]=fr[i]+sr[i];
                    f[i]=f[i]+width*g[i];
              }
        }
        for(i=1;i<=n;i++) f[i]=fr[i]/f[i];
        XPLOT(fr,n,Xstep,Ystep,Xpos,115,CYAN);
        XPLOT(f,n,Xstep,Ystep,Xpos,160,LIGHTGRAY);

        getch();
        free (a); free (sr); free (fr); free (g);free (f);
        closegraph();
}
```

Appendix 1.17

```
// twt&iwt2.cpp
// To demonstrate wavelet transform and its inverse by using
// constant relative bandwidth Gaussian windows.
// Author: M. D. Choudhry
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,Xpos,Xstep,Ystep,width;
        float *a,*g,*sr,*fr,*f,*si;

        a=(float *)calloc(n+1, sizeof(float));
        g=(float *)calloc(n+1, sizeof(float));
        sr=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(n+1, sizeof(float));
        f=(float *)calloc(n+1, sizeof(float));
        si=(float *)calloc(n+1, sizeof(float));

        Xpos=1; Xstep=2; Ystep=-30;
        INITIAL();SCRBLACK();

        TOPHAT(a,n,32);
        width=64;
        GAUSSCR(g,n,width,6);
        for(i=1;i<=n;i++) a[i]=30*width*g[i]*a[i];
        FFT1D(a,si,n,1);
        XPLOT(a,n,Xstep,Ystep,Xpos,70,GREEN);

        for(j=1;j<=6;j++){
              width=pow(2,j);
              GAUSSCR(g,n,width,2*j);
              XPLOT(g,n,Xstep,2*width*Ystep,1,70,MAGENTA);
              for(i=1;i<=n;i++){
                    sr[i]=width*g[i]*a[i];
                    fr[i]=fr[i]+sr[i];
                    f[i]=f[i]+width*g[i];
              }
        }
        for(i=1;i<=n;i++) f[i]=fr[i]/f[i];
        XPLOT(fr,n,Xstep,Ystep,Xpos,125,CYAN);
        XPLOT(f,n,Xstep,Ystep,Xpos,190,LIGHTGRAY);
        getch();

        free (a); free (sr); free (fr); free (g);free (f);free (si);
        closegraph();
}
```

Appendix 1.18

```
// twt&iwt3.cpp
// To demonstrate wavelet transform and its inverse by using
// constant relative bandwidth Gaussian windows.
// Author: M. D. Choudhry
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,Xpos,Xstep,Ystep,width;
        float *a,*g,*sr,*fr,*f,*si;

        a=(float *)calloc(n+1, sizeof(float));
        g=(float *)calloc(n+1, sizeof(float));
        sr=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(n+1, sizeof(float));
        f=(float *)calloc(n+1, sizeof(float));
        si=(float *)calloc(n+1, sizeof(float));

        Xpos=1; Xstep=2; Ystep=-30;
        INITIAL();SCRBLACK();

        COSINE(a,n,16);
        width=64;
        GAUSSCR(g,n,width,6);
        for(i=1;i<=n;i++) a[i]=20*width*g[i]*a[i];
        FFT1D(a,si,n,1);
        XPLOT(a,n,Xstep,Ystep,Xpos,70,GREEN);

        for(j=1;j<=6;j++){
              width=pow(2,j);
              GAUSSCR(g,n,width,2*j);
              XPLOT(g,n,Xstep,2*width*Ystep,1,70,MAGENTA);
              for(i=1;i<=n;i++){
                    sr[i]=width*g[i]*a[i];
                    fr[i]=fr[i]+sr[i];
                    f[i]=f[i]+width*g[i];
              }
        }
        for(i=1;i<=n;i++) f[i]=fr[i]/f[i];
        XPLOT(fr,n,Xstep,Ystep,Xpos,145,CYAN);
        XPLOT(f,n,Xstep,Ystep,Xpos,230,LIGHTGRAY);

        getch();
        free (a); free (sr); free (fr); free (g);free (f);free (si);
        closegraph();
}
```

Appendix 1.19

```
// wpacket1.cpp
// To demonstrate the formatiom of wavelet 4 packets consisting of
// (i) relative bandwidths (ii) constant relative bandwidths.
// Author: M. D. Choudhry
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>
#include "nrzozi.h"
#define n 64

void main(void)
{
        int i,j,p,nn,a;
        int width,Xstep,Ystep,Xpos,tstep;
        float *g1,*g2;

        g1=(float *)calloc(n+1, sizeof(float));
        g2=(float *)calloc(2*n+1, sizeof(float));

        Xpos=180; Xstep=2; Ystep=-20;
        INITIAL();SCRBLACK();

        for(j=1;pow(2,j)<=n/2;j++){
                width=pow(2,j);
                GAUSS11(g1,n,width,2*j);
                for(i=1;i<=n;i++) g2[i]=g1[n-i];
                NXPLOT(g2,n,Xstep,16*Ystep,Xpos,120,GREEN);
                NXPLOT(g2,n,Xstep,2*width*Ystep,Xpos,180,GREEN);
                NXPLOT(g2,n,Xstep,16*Ystep,Xpos,300,GREEN);
                NXPLOT(g2,n,Xstep,2*width*Ystep,Xpos,360,GREEN);
        }
        Xpos=180+(n-8)*Xstep;
        for(j=1;pow(2,j)<=n/2;j++){
                width=pow(2,j);
                GAUSS11(g1,n,width,2*j);
                NXPLOT(g1,n,Xstep,16*Ystep,Xpos,120,GREEN);
                NXPLOT(g1,n,Xstep,2*width*Ystep,Xpos,180,GREEN);
        }
        Xpos=180+(n-12)*Xstep;
        for(j=2;pow(2,j)<=n/2;j++){
                width=pow(2,j);
                GAUSS11(g1,n,width,2*j);
                NXPLOT(g1,n,Xstep,16*Ystep,Xpos,300,GREEN);
                NXPLOT(g1,n,Xstep,2*width*Ystep,Xpos,360,GREEN);
        }
        getch();
        closegraph();
        free (g1);free (g2);
}
```

Appendix 1.20

```
// twpacket.cpp
// To demonstrate the formatiom of wavelet packet consisting of
// (i)  relative bandwidths (ii) constant relative bandwidths.
// Author: M. D. Choudhry
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>
#include "nrzozi.h"
#define n 64

void main(void)
{
        int i,j,p,nn,a;
        int width,Xstep,Ystep,Xpos,tstep;
        float *g1,*g2,*f,*fr,*sr;

        f=(float *)calloc(2*n+1, sizeof(float));
        g1=(float *)calloc(n+1, sizeof(float));
        g2=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(2*n+1, sizeof(float));
        sr=(float *)calloc(n+1, sizeof(float));

        Xpos=180; Xstep=2; Ystep=-20;
        INITIAL();SCRBLACK();

        COSINE(f,2*n,8);NXPLOT(f,2*n,Xstep,Ystep,Xpos,200,GREEN);
        for(j=1;pow(2,j)<=n/2;j++){
                width=pow(2,j);
                GAUSSCR(g1,n,width,2*j);
                for(i=1;i<=n;i++){
                    sr[i]=width*f[i]*g1[i];
                    fr[i]=fr[i]+sr[i];
                    sr[i]=width*f[i]*g1[n-i];
                    fr[n+i]=fr[n+i]+sr[i];
                    g2[i]=g1[n-i];
                }
                Xpos=180;
                NXPLOT(g1,n,Xstep,16*Ystep,Xpos,140,MAGENTA);
                NXPLOT(g1,n,Xstep,2*width*Ystep,Xpos,200,MAGENTA);
                Xpos=180+(n-1)*Xstep;
                NXPLOT(g2,n,Xstep,16*Ystep,Xpos,140,MAGENTA);
                NXPLOT(g2,n,Xstep,2*width*Ystep,Xpos,200,MAGENTA);
        }
        Xpos=180;
        NXPLOT(fr,2*n,Xstep,Ystep,Xpos,290,LIGHTGRAY);
        getch();
        closegraph();
        free (g1);free (g2);
}
```

Appendix 1.21

```
// wpacket3.cpp
// To demonstrate the formatiom of a double density wavelet packet
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>
#include "nrzozi.h"
#define n 64

void main(void)
{
        int i,j,p,nn,a;
        int width,Xstep,Ystep,Xpos,tstep;
        float *g1,*g2,*f,*fr,*sr;

        g1=(float *)calloc(n+1, sizeof(float));
        g2=(float *)calloc(n+1, sizeof(float));
        f=(float *)calloc(n+1, sizeof(float));
        fr=(float *)calloc(n+1, sizeof(float));
        sr=(float *)calloc(n+1, sizeof(float));

        Xpos=180; Xstep=4; Ystep=-20;
         INITIAL();
         SCRBLACK();

        COSINE(f,n,8);
        NXPLOT(f,n,Xstep,Ystep,Xpos,220,GREEN);
        for(j=1;j<=6;j++){
                width=pow(2,j);
                GAUSSCR(g1,n,width,2*j);
                for(i=1;i<=n;i++){
                        sr[i]=width*f[i]*g1[i];
                        fr[i]=fr[i]+sr[i]/2;
                        g2[i]=g1[n-i];
                        sr[i]=width*f[i]*g2[i]/2;
                        fr[i]=fr[i]+sr[i];
                }
                NXPLOT(g1,n,Xstep,16*Ystep,Xpos,160,MAGENTA);
                NXPLOT(g2,n,Xstep,16*Ystep,Xpos,160,MAGENTA);
                NXPLOT(g1,n,Xstep,2*width*Ystep,Xpos,220,MAGENTA);
                NXPLOT(g2,n,Xstep,2*width*Ystep,Xpos,220,MAGENTA);
        }
        NXPLOT(fr,n,Xstep,Ystep,Xpos,290,LIGHTGRAY);
        getch();

        closegraph();
        free (g1);free (g2);free (f);free (fr);free (sr);
}
```

Appendix 1.22

```
// tcrbbcg.cpp
// To test and demonstrate the formulation of Constant
// Relative Bandwidth Binary Gaussian six windows.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>

#include "nrzozi.h"

#define n 32

void main(void)
{
   int i,j,*s,Xpos,Xstep,Ystep,width;
   float *g;

   s=(int *)calloc(n+1, sizeof(int));
   g=(float *)calloc(n+1, sizeof(float));

   Xpos=1;
   Xstep=8;
   Ystep=-16;

   INITIAL();
   SCRBLACK();

   for(j=1;j<=6;j++){
     width=pow(2,j);
     GAUSSCR(g,n,width,j);
     XPLOT(g,n,Xstep,2*(2*width*Ystep),Xpos,60,MAGENTA);
     for(i=0;i<n;i++){
        g[i]=2*width*g[i];
        if(g[i]<.3737)s[i]=-1;
        else if(g[i]<.7474)s[i]=0;
        else if(g[i]<1.1284)s[i]=1;
     }
     XPLOTI(s,n,Xstep,Ystep,Xpos,j*60+60,MAGENTA);
   }

   getch();
   closegraph();

   free (s);
   free (g);

}
```

Appendix 1.23

```
// twtcrbcg.cpp
// Test Wavelet Transform and Inverse using Constant Relative Binary Coded Gaussian windows.
// Author: M. D. Choudhry
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "nrzozi.h"
#define n 64

void main(void)
{
   int i,j,*s,Xpos,Xstep,Ystep,width;
   float *a,*g,*fr,*f;
   FILE *fp;

   s=(int *)calloc(n+1, sizeof(int));
   a=(float *)calloc(n+1, sizeof(float));
   g=(float *)calloc(n+1, sizeof(float));
   fr=(float *)calloc(n+1, sizeof(float));
   f=(float *)calloc(n+1, sizeof(float));
   if((fp=fopen("hw.dat", "wb"))==NULL) { printf("Cannot open output file.\n");exit(1);}

   Xpos=1;Xstep=4;Ystep=-16;
   INITIAL();SCRBLACK();
   COSINE(a,n,4);XPLOT(a,n,Xstep,Ystep,Xpos,70,GREEN);
   for(j=1;j<=6;j++){
     width=pow(2,j);
     GAUSSCR(g,n,width,j); XPLOT(g,n,Xstep,2*width*Ystep,Xpos,70,MAGENTA);
     for(i=0;i<n;i++){
        g[i]=2*width*g[i];
        if(g[i]<.3737)s[i]=-1;
        else if(g[i]<.7474)s[i]=0;
        else if(g[i]<1.1284)s[i]=1;
     }
     XPLOTI(s,n,Xstep,Ystep,Xpos,140,MAGENTA);
     for(i=0;i<n;i++){
       fr[i]=fr[i]+s[i]*a[i];
       f[i]=f[i]+s[i];
     }
     for(i=0;i<n;i++)fprintf(fp,"%3d",s[i]);
     fprintf(fp,"\n");
   }
   for(i=0;i<n;i++)f[i]=fr[i]/f[i];
   XPLOT(fr,n,Xstep,Ystep/4,Xpos,210,CYAN);
   XPLOT(f,n,Xstep,Ystep,Xpos,280,LIGHTGRAY);
   getch();
   closegraph();
   free (s);free (a); free (fr); free (g);free (f); fclose(fp);
}
```

Appendix 1.24

```
// tmrbcgwt.cpp
// Testing MultiResolution Binary Coded Gaussian(windows) Wavelet Transform
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "nrzozi.h"
#define n 128

void main(void)
{
   int *g,i,j,k,nn,r,a,wwp,width,Xstep,Ystep,Xpos;
   float *f,*fo,*nf,sum,eng;

   g=(int *)calloc(n+1, sizeof(int));
   f=(float *)calloc(n+1, sizeof(float));
   nf=(float *)calloc(2*n+1, sizeof(float));
   fo=(float *)calloc(n+1, sizeof(float));

   wwp=8;Xpos=1;Xstep=2;Ystep=-2;
   INITIAL();
   SCRWHITE();

   COSINE(f,n,8);
   for(i=0;i<=n;i++)f[i]=3*f[i];
   RNOISE(f,nf,n,10);XPLOT(nf,n,Xstep,Ystep,Xpos,80,BLUE);

   for(j=0;j<=2;j++){
      a=pow(2,j);
      nn=n/a;
      for(i=0;i<=n;i++)fo[i]=0;
      for(k=0;k<nn;k++){
         sum=0.;eng=0.;
         for(r=1;r<=6;r++){
            width=pow(2,r);
            BCGAUSS(g,n,width,r);
            XPLOTI(g,wwp,Xstep,2*Ystep,280,j*80+150,BLUE);
            for(i=1;i<=wwp;i++){
               sum=sum+g[i]*nf[(i-1)+a*k];
               eng=eng+g[i];
            }
         }fo[k]=sum/eng;
      }XPLOT(fo,n/a,Xstep,2*Ystep,Xpos,j*80+150,BLUE);
   }
    getch();
   closegraph();
   free (f);free (f);free (nf);free (fo);
}
```

Appendix 2

Appendix 2.1

```
// tawczot.cpp
// To test/demonstrate automatic wavelet compressing
// zoom-out wavelet transform which reduces noise as well.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,k,nn,a,m=16;
        int width,Xstep,Ystep,Xpos,tstep;
        float *f,*w,*fo,*nf,sum;

        f=(float *)calloc(n+1, sizeof(float));
        w=(float *)calloc(n+1, sizeof(float));
        fo=(float *)calloc(n+1, sizeof(float));
        nf=(float *)calloc(n+1, sizeof(float));

        Xpos=1;
        Xstep=2;
        Ystep=-20;

        INITIAL();
        SCRBLACK();
        COSINE(f,n,8);
        RNOISE(f,nf,n,2);

        for(j=0;j<=2;j++){
                a=pow(2,j);
                for(i=0;i<=n;i++)fo[i]=0.;
                AWCZOT(nf,fo,n/a,m,-j);
                XPLOT(nf,n/a,Xstep,Ystep,Xpos,j*100+50,GREEN);
                XPLOT(fo,n,Xstep,(j+1)*Ystep/6,Xpos,j*100+100,LIGHTGRAY);
        }
        getch();

        closegraph();
        free (f);
        free (w);
        free (fo);
        free (nf);
}
```

Appendix 2.2

```
// tawezit.cpp
// To test/demonstrate automatic wavelet expanding zoom-in
// wavelet transform which reduces noise as well.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,k,nn,a,m=8;
        int width,Xstep,Ystep,Xpos,tstep;
        float *f,*w,*fo,*nf,sum,lemda;

        f=(float *)calloc(n+1, sizeof(float));
        w=(float *)calloc(n+1, sizeof(float));
        fo=(float *)calloc(n+1, sizeof(float));
        nf=(float *)calloc(n+1, sizeof(float));

        Xpos=1;
        Xstep=2;
        Ystep=-20;

        INITIAL();
        SCRBLACK();

        COSINE(f,n,4);
        RNOISE(f,nf,n,2);
        XPLOT(nf,n,Xstep,Ystep,Xpos,60,GREEN);

        for(j=0;j<=2;j++){
                a=pow(2,j);
                for(i=0;i<=n;i++)fo[i]=0.;
                AWEZIT(nf,fo,n,m,j);
                XPLOT(fo,n/a,Xstep,Ystep/6,1,(j+1)*50+70,LIGHTGRAY);
        }
        getch();
        closegraph();

        free (f);
        free (w);
        free (fo);
        free (nf);

}
```

Appendix 3

Appendix 3.1

```
//
// tnrzo.cpp
// Testing of Noise Reduction and Zoom-out wavelet transform
// which uses a single wavelet and keeps its size the fixed.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,a,Xstep,Ystep,Xpos;m;
        float *f,*fo,*nf,sum;

        f=(float *)calloc(n+1, sizeof(float));
        fo=(float *)calloc(n+1, sizeof(float));
        nf=(float *)calloc(n+1, sizeof(float));

        Xpos=1;
        Xstep=2;
        Ystep=-10;

        INITIAL();
        SCRBLACK();

        COSINE(f,n,8);
        RNOISE(f,nf,n,2);

        m=16;
        for(j=0;j<=2;j++){
                a=pow(2,j);
                for(i=0;i<=n;i++)fo[i]=0.;
                NRZO(nf,fo,n/a,m,-j);
                XPLOT(nf,n/a,Xstep,Ystep,Xpos,j*100+50,GREEN);
                XPLOT(fo,n,Xstep,Ystep/4*(j+1),Xpos,j*100+100,LIGHTGRAY);
        }
        getch();

        closegraph();
         free (f);
         free (fo);
         free (nf);

}
```

Appendix 3.2

```
// tnr.cpp
// Testing of Noise Rducing wavelet transform
// which uses a single wavelet and keeps its size the fixed.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,k,m,Xstep,Ystep,Xpos;
        float *f,*fo,*nf,sum;

        f=(float *)calloc(n+1, sizeof(float));
        fo=(float *)calloc(2*n+1, sizeof(float));
        nf=(float *)calloc(n+1, sizeof(float));

        Xpos=1; Xstep=2; Ystep=-40;
        INITIAL();
        SCRBLACK();

        COSINE(f,n,8);
        RNOISE(f,nf,n,2); XPLOT(nf,n,Xstep,Ystep,Xpos,70,GREEN);

        for(j=2;j<=4;j++){
              m=pow(2,j);
              for(i=0;i<=n;i++)fo[i]=0.;
              NR(nf,fo,n,m);XPLOT(fo,n,Xstep,Ystep,Xpos,j*100,LIGHTGRAY);
        }
        getch();

        closegraph();
        free (f);free (fo);free (nf);
}
```

Appendix 3.3

```
// tnrzi.cpp
// To demonstrate a wavelet transform which reduces
// noise and compresses data simultaneously.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
```

```
#include <conio.h>
#include <alloc.h>
#include <math.h>
#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,k,nn,a,m;
        int width,Xstep,Ystep,Xpos,tstep;
        float *f,*fo,*nf,sum;

        f=(float *)calloc(n+1, sizeof(float));
        fo=(float *)calloc(n+1, sizeof(float));
        nf=(float *)calloc(n+1, sizeof(float));

        Xpos=1;
        Xstep=2;
        Ystep=-10;

        INITIAL();
        SCRBLACK();

        COSINE(f,n,4);
        RNOISE(f,nf,n,2);
        XPLOT(nf,n,Xstep,Ystep,Xpos,60,GREEN);

        m=16;
        for(j=0;j<=2;j++){
                a=pow(2,j);
                for(i=0;i<=n;i++)fo[i]=0.;
                NRZI(nf,fo,n,m,j);
                XPLOT(fo,n/a,Xstep,Ystep/4,Xpos,(j+1)*50+70,LIGHTGRAY);
        }

        for(i=0;i<=n;i++){f[i]=0.;nf[i]=0.;}

        TOPHAT(f,n,32);
        RNOISE(f,nf,n,2);
        XPLOT(nf,n,Xstep,Ystep,Xpos,280,GREEN);

        for(j=0;j<=2;j++){
                a=pow(2,j);
                for(i=0;i<=n;i++)fo[i]=0.;
                NRZI(nf,fo,n,32,j);
                XPLOT(fo,n/a,Xstep,Ystep/6,Xpos,(j+1)*50+280,LIGHTGRAY);
        }
        getch();
        closegraph();
        free (f);
        free (fo);
        free (nf);

}
```

Appendix 3.4

```
// tzoomout.cpp
// To demonstrate a wavelet transform which
// expands data without smoothing it.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,a;
        int width,Xstep,Ystep,Xpos;
        float *f,*nf,*zo;

        f=(float *)calloc(n+1, sizeof(float));
        nf=(float *)calloc(n+1, sizeof(float));
        zo=(float *)calloc(n+1, sizeof(float));

        Xpos=1;
        Xstep=2;
        Ystep=-10;

        INITIAL();
        SCRBLACK();
        COSINE(f,n,8);
        RNOISE(f,nf,n,2);
        for(j=1;j<=3;j++){
                a=pow(2,j);
                for(i=0;i<=n;i++)zo[i]=0.;
                ZOOMOUT(f,zo,n/a,j);
                XPLOT(f,n/a,Xstep,Ystep,Xpos,j*70-40,GREEN);
                XPLOT(zo,n,Xstep,Ystep,Xpos,j*70-10,LIGHTGRAY);
        }
        for(j=1;j<=3;j++){
                a=pow(2,j);
                for(i=0;i<=n;i++)zo[i]=0.;
                ZOOMOUT(nf,zo,n/a,j);
                XPLOT(nf,n/a,Xstep,Ystep,Xpos,j*70+180,GREEN);
                XPLOT(zo,n,Xstep,Ystep,Xpos,j*70+210,LIGHTGRAY);
        }
        getch();

        closegraph();

        free (f);free (nf);free (zo);
}
```

Appendix 3.5

```
// tremann.cpp
// Testing of the remarkable ANN classifier developed in Section(5.9)
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <alloc.h>
#include <math.h>
#include "nrzozi.h"
#define n 128

void main(void)
{
   int Xstep,Ystep,Xpos;
   int i,j,k,su,sd,mir,zw,xaz,nc;

   float lemda,net,eps;
   float *w,*x,*xa,*xp,*r,*delta;

   x=(float *)calloc(n+1, sizeof(float));
   w=(float *)calloc(n+1, sizeof(float));
   xa=(float *)calloc(n+1, sizeof(float));
   xp=(float *)calloc(n+1, sizeof(float));
   r=(float *)calloc(n+1, sizeof(float));
   delta=(float *)calloc(n+1, sizeof(float));

   Xpos=1;
   Xstep=2;
   Ystep=-8;

   INITIAL();
   SCRBLACK();

   COSINE(x,n,8);
   XPLOT(x,n,Xstep,Ystep,Xpos,100,GREEN);

   // construct weights for cosine
   lemda=0.;
   for(i=0;i<n;i++)lemda=lemda+x[i]*x[i];
   for(i=0;i<n;i++)w[i]=x[i]/lemda;

   // input layer
   //for(i=0;i<n;i++)xa[i]=x[i]; //The known pattern
   //for(i=0;i<n;i++)xa[i]=5*x[i];//Scaled up version of known pattern
   //for(i=0;i<n;i++)xa[i]=x[i]/5.;//Scaled down version of known pattern
   //for(i=0;i<=n;i++)xa[i]=-5*x[i];//Mirror image(scaled up) of known pattern
   for(i=0;i<=n;i++)x[i]=3*x[i];// To keep noise/signal ratio small
   RNOISE(x,xa,n,10);// Noisy version of known pattern

   XPLOT(xa,n,Xstep,Ystep,Xpos,200,LIGHTGRAY);
```

```
// hidden layer1
net=0.;
for(i=0;i<n;i++)net=net+w[i]*xa[i];
if(net>.95 && net<1.05){
        for(i=0;i<n;i++)xa[i]=lemda*w[i];
        printf("\nThe pattern is known !.\n");
        XPLOT(xa,n,Xstep,Ystep,Xpos,120,LIGHTGRAY);
}

else { //hidden layer2
su=0;sd=0;mir=0;zw=0,xaz=1;
for(k=0;k<n;k++){
        if(w[k]!=0.){
                r[k]=xa[k]/(lemda*w[k]);
                delta[k]=abs(lemda*w[k]-xa[k]);
                if(xa[k]==0.)xaz=xaz+1;
                if(r[k]<0.)mir=mir+1;
                else if(r[k]>0.&& r[k]<1.)sd=sd+1;
                else if(r[k]>1.)su=su+1;
        }else zw=zw+1;
}
if(su>n-zw-2)printf("\nScaled up version of the sample.!\n");
else if(sd>n-zw-xaz)printf("\nScaled down version of the sample.!\n");
else if(mir>n-zw-xaz)printf("\nMirror image (scaled up) of the sample.!\n");
else printf("\nNoisy pattern! hidden layer3 is activated.\n");
}

// hidden layer3
FDBPB(x,xp,n);XPLOT(xp,n,Xstep,Ystep,Xpos,260,LIGHTGRAY);
eps=1.;
for(k=0;k<n;k++){
        if(delta[k]<eps*xp[k])nc=nc+1;
}
if(nc>n-2)printf("\nNoise is removable!\n");
else printf("\n Purely unknown or corrupted badely!\n");

getch();

closegraph();

free (x);
free (w);
free (xa);
free (xp);
free (r);
free (delta);

}
```

Appendix 3.6

```
// tfsegs.cpp
// Test of finding segments of thresholds patterns.
// Author: M. D. Choudhry
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,k,t,t1;
        float c,maxy,miny,eps,delta,Xstep,Ystep,Xpos;
        float *f,*y,*fo,d1,d2;

        f=(float *)calloc(n+1, sizeof(float));
        y=(float *)calloc(n+1, sizeof(float));
        fo=(float *)calloc(n+1, sizeof(float));

        Xpos=1;Xstep=2;Ystep=-4;
        INITIAL();SCRBLACK();
        COSINE(f,n,4);for(i=0;i<n;i++)y[i]=4*f[i];
        XPLOT(y,n,Xstep,Ystep,Xpos,100,GREEN);
        eps=1.99; c=0.2;
        maxy=y[0];miny=y[0];
        for(i=1;i<n;i++){
                delta=0.; t1=i-1;
                while(delta<=eps && i<n){
                        i=i+1;
                        d1=y[i]-y[i-1];
                        d2=y[i+1]-y[i];
                        if(d1>=0.){
                                if((d2-d1)<=0.)maxy=y[i];
                                else miny=y[i-1];
                        }
                        else {
                                if((d2-d1)<=0.)miny=y[i-1];
                                else maxy=y[i-1];
                        }
                        delta=abs(maxy-miny);
                }
                t=i-t1;
                for(j=1;j<=t;j++)fo[t1+j]=(1+c)*maxy;
                miny=y[i];maxy=y[i];
        }
        XPLOT(fo,n,Xstep,Ystep,Xpos,100,LIGHTGRAY);
        getch();
        closegraph();
        free (f);free (y); free (fo);
}
```

Appendix 3.7

```
// BT.CPP
// Binary Transform of a Signal
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>

#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j,k,t,t1;
        float c,v,maxy,miny,delta,Xstep,Ystep,Xpos;
        float *f,*y,*seg,*bt,d1,d2;

        f=(float *)calloc(n+1, sizeof(float));
        y=(float *)calloc(n+1, sizeof(float));
        seg=(float *)calloc(n+1, sizeof(float));
        bt=(float *)calloc(n+1, sizeof(float));

        Xpos=1;Xstep=2;Ystep=-4;
        INITIAL();SCRBLACK();
        COSINE(f,n,1);for(i=0;i<n;i++)y[i]=5*f[i];
        XPLOT(y,n,Xstep,Ystep,Xpos,100,GREEN);

        v=.5;
        c=0.2;
        maxy=y[0];miny=y[0];
        for(i=1;i<n;i++){
                delta=0.;
                t1=i-1;

                // Detect a segment, find its length 't' and biggest eltement 'maxy'
                while(delta<=v && i<n){
                        i=i+1;
                        d1=y[i]-y[i-1];
                        d2=y[i+1]-y[i];
                        if(d1>=0.){
                             if((d2-d1)<=0.)maxy=y[i];
                             else miny=y[i-1];
                        }
                        else {
                              if((d2-d1)<=0.)miny=y[i-1];
                              else maxy=y[i-1];
                        }
                        delta=abs(maxy-miny);
                }
                t=i-t1;
```

```
            // Produce the horizontal segment passing through 'maxy'
            for(j=1;j<=t;j++)seg[t1+j]=(1+c)*maxy;

            // Compute BT
            for(j=1;j<=t;j++){
                  d1=y[t1+j]-y[t1+j+1];
                  if(d1>=0 && d1<=v/3)bt[t1+j]=1;
                  else if(d1>v/3 && d1<(2*v)/3)bt[t1+j]=0;
                  else if(d1>=(2*v)/3 && d1<=v)bt[t1+j]=-1;
            }
            miny=y[i];maxy=y[i];
      }
      XPLOT(seg,n,Xstep,Ystep,Xpos,100,LIGHTGRAY);
      XPLOT(y,n,Xstep,Ystep,Xpos,160,GREEN);
      XPLOT(bt,n,Xstep,5*Ystep,Xpos,160,LIGHTGRAY);

      getch();
      closegraph();
      free (f);free (y); free (seg);free (bt);
}
```

Appendix 4

Appendix(4.1) (A Data Smoothing Program)

In this program the wavelet $\frac{1}{4}\{1,2,1\}$ is applied.

```
// smoth1d1.cpp to smooth image data horizontally.
// Author: M. D. Choudhry
//
#include <stdio.h>
#include <stdlib.h>
#define n 128

void main()
{
        int i,j;
        int **a,**b;
        FILE *fp1,*fp2;

        a=new int *[n+1];
        for(j=0;j<n+1;j++)a[j]=new int [n+1];
        b=new int *[n+1];
        for(j=0;j<n+1;j++)b[j]=new int [n+1];

        if((fp1=fopen("mona1.dat", "rb"))==NULL) {
                printf("Cannot open input file.\n");exit(1);}
        if((fp2=fopen("mona2.dat", "wb"))==NULL) {
                printf("Cannot open input file.\n");exit(1);}

        for(i=0;i<n;i++){
                for(j=0;j<n;j++)fscanf(fp1,"%4d",&b[i][j]);
        }

        for(i=0;i<=n-2;i++){
                a[i][0]=(2*b[i][0]+b[i][1])/3;
                for(j=1;j<=n-2;j++){
                        a[i][j]=(b[i][j-1]+2*b[i][j]+b[i][j+1])/4;
                }
                a[i][n-1]=(b[i][n-2]+2*b[i][n-1])/3;
        }

        for(i=0;i<n;i++){
                for(j=0;j<n;j++)fprintf(fp2,"%4d",a[i][j]);fprintf(fp2,"\n");
        }

        fclose(fp1);
         fclose(fp2);
        for(j=0;j<n;j++)delete[] a[j];
        delete[] a;
        for(j=0;j<n;j++)delete[] b[j];
        delete[] b;
}
```

Appendix(4.2) (A Data Smoothing Program)

In this program the wavelet $\frac{1}{2}\left\{\frac{1}{8},\frac{2}{8},\frac{3}{8},\frac{4}{8},\frac{3}{8},\frac{2}{8},\frac{1}{8}\right\}$ is applied.

```
// smoth1d2.cpp to smooth image data horizontally.
// Author: M. D. Choudhry
#include <stdio.h>
#include <stdlib.h>
#define n 128

void main()
{
        int i,j;
        int **a,**b;
        FILE *fp1,*fp2;

        a=new int *[n+1];
        for(j=0;j<n+1;j++)a[j]=new int [n+1];
        b=new int *[n+1];
        for(j=0;j<n+1;j++)b[j]=new int [n+1];

        if((fp1=fopen("mona1.dat", "rb"))==NULL) {printf("Cannot open input file.\n");exit(1);}

        if((fp2=fopen("mona2.dat","wb"))==NULL){printf("Cannot open input file.\n");exit(1);}

        for(i=0;i<n;i++){
                for(j=0;j<n;j++)fscanf(fp1,"%4d",&b[i][j]);
        }

        for(i=0;i<n;i++){
                a[i][0]=(4*b[i][0]+3*b[i][1]+2*b[i][2]+b[i][3])/10;
                a[i][1]=(3*b[i][0]+4*b[i][1]+3*b[i][2]+2*b[i][3]+b[i][4])/13;

        a[i][2]=(2*b[i][0]+3*b[i][1]+4*b[i][2]+3*b[i][3]+2*b[i][4]+b[i][5])/15;
                for(j=3;j<=124;j++){
                        a[i][j]=(b[i][j-3]+2*b[i][j-2]+3*b[i][j-1]+4*b[i][j]+3*b[i][j+1]
                                +2*b[i][j+2]+b[i][j+3])/16;
                }
                a[i][125]=(b[i][122]+2*b[i][123]+3*b[i][124]
                        +4*b[i][125]+3*b[i][126]+2*b[i][127])/15;
                a[i][126]=(b[i][123]+2*b[i][124]+3*b[i][125]
                        +4*b[i][126]+3*b[i][127])/13;
                a[i][127]=(b[i][124]+2*b[i][125]+3*b[i][126]+4*b[i][127])/10;
        }
        for(i=0;i<n;i++){
                for(j=0;j<n;j++)fprintf(fp2,"%4d",a[i][j]);fprintf(fp2,"\n");
        }
        fclose(fp1); fclose(fp2);
        for(j=0;j<n;j++)delete[] a[j]; delete[] a;
        for(j=0;j<n;j++)delete[] b[j]; delete[] b;
}
```

Appendix(4.3) (A Two Dimensional Data Smoothing Program)

```
// smoth2d1.cpp
// The wavelet given in relation(9.3.1) is implemented in this program.
// Author: M. D. Choudhry
#include <stdio.h>
#include <stdlib.h>

void main()
{
        int i,j,**a,**b;
        FILE *fp1;

        a=new int *[129];
        for(j=0;j<129;j++)a[j]=new int [129];
        b=new int *[129];
        for(j=0;j<129;j++)b[j]=new int [129];
        if((fp1=fopen("mona2.dat", "r+"))==NULL) { printf("Cannot open input ile.\n");exit(1);}

        // Read a file to the array b[][].
        for(i=0;i<128;i++){
                for(j=0;j<128;j++)fscanf(fp1,"%4d",&b[i][j]);
        }
        // process first row of the image
        a[0][0]=(2*b[0][0]+b[0][1])/3;
        for(j=1;j<=126;j++)a[0][j]=(b[0][j-1]+2*b[0][j]+b[0][j+1])/4;
        a[0][127]=(b[0][126]+2*b[0][127])/3;

        // Process all rows except the first and the last row.
        for(i=1;i<=126;i++){
                a[i][0]=(b[i-1][0]+2*b[i][0]+b[i+1][0]+b[i-1][1]+b[i][1]+b[i+1][1])/7;
                for(j=1;j<=126;j++){
                        a[i][j]=(b[i-1][j-1]+b[i][j-1]+b[i+1][j-1]+b[i1][j]+2*b[i][j]+b[i+1][j]
                                +b[i-1][j+1]+b[i][j+1]+b[i+1][j+1])/10;
                }
                a[i][127]=(b[i-1][126]+b[i][126]+b[i+1][126]
                        +b[i-1][127]+2*b[i][127]+b[i+1][127])/7;
        }

        // Process the last row
        a[127][0]=(2*b[127][0]+b[127][1])/3;
        for(j=1;j<=126;j++)a[127][j]=(b[127][j-1]+2*b[127][j]+b[127][j+1])/4;
        a[127][127]=(b[127][126]+2*b[127][127])/4;

        if(fseek(fp1,0,SEEK_SET)){printf("Seek error.|n");exit(1);}
        for(i=0;i<128;i++){
                for(j=0;j<128;j++)fprintf(fp1,"%4d",a[i][j]);fprintf(fp1,"\n");
        }
        fclose(fp1);
        for(j=0;j<129;j++)delete[] a[j];
        delete[] a;
        for(j=0;j<129;j++)delete[] b[j];
        delete[] b;
}
```

Appendix(4.4) (A Program for Data Compression Plus Noise Reduction)

```
// twave22.cpp
// Author: M. D. Choudhry
//
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "img.h"
#define n 128

void main()
{
        int i,j,k,a,p,q,r=1;
        FILE *fp1,*fp2;
        float *s1,*s2,**arr;

        s1=new float [n+1];
        s2=new float [n+1];
        arr=new float *[n+1];
        for(j=0;j<n+1;j++)arr[j]=new float [n+1];

        if((fp1=fopen("mona1.dat", "rb"))==NULL){printf("Cannot open input file.\n");exit(1);}
        if((fp2=fopen("check.dat","wb"))==NULL){printf("Cannot open output file.\n");exit(1);}

        // Read the image "mona1.dat" into the array "arr[128][129]"
        for(i=0;i<n;i++){
                for(j=0;j<n;j++)fscanf(fp1,"%4f",&arr[i][j]);
        }

        //Compress the image horizontally by applying 1D wavelet transform
        for(i=0;i<n;i++){
                for(j=0;j<n;j++)s1[j]=arr[i][j];
                for(j=0;j<=r;j++){
                        a=powl(2,j);
                        WAVE(s1,s2,n/a,j,1);
                        for(k=0;k<n/(2*a);k++)s1[k]=s2[k];
                }
                for(j=0;j<n/(2*a);j++)arr[i][j]=s1[j];
        }
        q=n/(2*a);

        //Compute inverse wavelet transform
        for(i=0;i<n;i++){
                for(j=0;j<q;j++)s2[j]=arr[i][j];
                for(j=0;j<=r;j++){
                        a=pow(2,r-j);
                        WAVE(s2,s1,n/(2*a),j,-1);
                        for(k=0;k<n/a;k++)s2[k]=s1[k];
                }
                for(j=0;j<n/a;j++)arr[i][j]=s1[j];
        }
```

```
        // Save the reconstructed image in the file pointed by "fp2"
        for(i=0;i<n;i++){
                for(k=0;k<8;k++){
                        p=16*k;
                        for(j=0;j<16;j++)fprintf(fp2,"%4d",(int)arr[i][j+p]);
                        fprintf(fp2,"\n");
                }
        }
        fclose(fp1);
        fclose(fp2);
        delete []s1;
        delete []s2;
        for(j=0;j<n+1;j++)delete []arr[j];
        delete []arr;
}
```

Appendix(4.5) (One-dimensional Wavelet applied to two-dimensional Data)

```
// twave222.cpp
// Author: M. D. Choudhry
//
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "img.h"
#define n 128

void main()
{
        int i,j,k,a,p,q,r=1;
        FILE *fp1,*fp2;
        float *s1,*s2,**arr;

        s1=new float [n+1];
        s2=new float [n+1];
        arr=new float *[n+1];
        for(j=0;j<n+1;j++)arr[j]=new float [n+1];
        if((fp1=fopen("mona1.dat", "rb"))==NULL){printf("Can't open input file!\n");exit(1);}
        if((fp2=fopen("twave222.dat","wb"))==NULL){printf("Can't open out file!\n");exit(1);}

        // Read mona1.dat into the array "arr[n][n]"
        for(i=0;i<n;i++){
                for(j=0;j<n;j++)fscanf(fp1,"%4f",&arr[i][j]);
        }
        // Perform data compressing wavelet transform column-by-column
        for(i=0;i<n;i++){
                for(j=0;j<n;j++)s1[j]=arr[j][i];
                for(j=0;j<=r;j++){
                        a=powl(2,j);
                        WAVE(s1,s2,n/a,j,1);
                        for(k=0;k<n/(2*a);k++)s1[k]=s2[k];
                }
```

```
            for(j=0;j<n/(2*a);j++)arr[j][i]=s1[j];
        }
        q=n/(2*a);

        //Apply row-wise the wavelet transform on the column-wise compressed image
        for(i=0;i<q;i++){
            for(j=0;j<n;j++)s1[j]=arr[i][j];
            for(j=0;j<=r;j++){
                a=powl(2,j);
                WAVE(s1,s2,n/a,j,1);
                for(k=0;k<n/(2*a);k++)s1[k]=s2[k];
            }
            for(j=0;j<n/(2*a);j++)arr[i][j]=s1[j];
        }

        //Apply the inverse wavelet transform column-wise
        for(i=0;i<q;i++){
            for(j=0;j<q;j++)s2[j]=arr[i][j];
            for(j=0;j<=r;j++){
                a=pow(2,r-j);
                WAVE(s2,s1,q/(2*a),j,-1);
                for(k=0;k<q/a;k++)s2[k]=s1[k];
            }
            for(j=0;j<q/a;j++)arr[i][j]=s1[j];
        }


        //Apply the inverse wavelet transform row-wise
        for(i=0;i<n;i++){
            for(j=0;j<q;j++)s2[j]=arr[j][i];
            for(j=0;j<=r;j++){
                a=pow(2,r-j);
                WAVE(s2,s1,n/(2*a),j,-1);
                for(k=0;k<n/a;k++)s2[k]=s1[k];
            }
            for(j=0;j<n/a;j++)arr[j][i]=s1[j];
        }
        // Save the reconstruted image as a binary file pointed by the "fp2"
        for(i=0;i<n;i++){
            for(k=0;k<8;k++){
                p=16*k;
                for(j=0;j<16;j++)fprintf(fp2,"%4d",(int)arr[i][j+p]);
                fprintf(fp2,"\n");
            }
        }

        fclose(fp1);
        fclose(fp2);
        delete []s1;
        delete []s2;
        for(j=0;j<n+1;j++)delete []arr[j];
        delete []arr;

}
```

Appendix(4.6) (Using Two-dimensional Wavelet and its Inverse Wavelet for Data Compression Plus Noise Reduction)

```
// twave23.cpp
// Author: M. D. Choudhry
//
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include "img.h"
#define n 128

void main()
{
        int i,j,k,a,p,q,r=1;
        FILE *fp1,*fp2,*fp3;
        float **arr1,**arr2,*s1,*s2;

        s1=new float [n+1];
        s2=new float [n+1];
        arr1=new float *[n+1];
        for(j=0;j<n+1;j++)arr1[j]=new float [n+1];
        arr2=new float *[n+1];
        for(j=0;j<n+1;j++)arr2[j]=new float [n+1];

        if((fp1=fopen("mona1.dat", "rb"))==NULL) {
                printf("Cannot open input file.\n");exit(1);}
        if((fp2=fopen("twave23c.dat", "wb"))==NULL) {
                printf("Cannot open output file.\n");exit(1);}
        if((fp3=fopen("twave23r.dat", "wb"))==NULL) {
                printf("Cannot open output file.\n");exit(1);}

        for(i=0;i<n;i++){
                for(j=0;j<n;j++)fscanf(fp1,"%4f",&arr1[i][j]);
        }

        for(j=0;j<=r;j++){
                a=powl(2,j);
                WAVE23(arr1,arr2,n/a,1);
                for(i=0;i<n/(2*a);i++){
                        for(k=0;k<n/(2*a);k++)arr1[i][k]=arr2[i][k];
                }
        }

        q=n/(2*a);

        for(i=0;i<n;i++){
                for(k=0;k<8;k++){
                        p=16*k;
                        for(j=0;j<16;j++)fprintf(fp2,"%4d",(int)arr1[i][j+p]);fprintf(fp2,"\n");
                }
        }
```

```
        for(i=q;i<n;i++){
                for(k=0;k<n;k++){arr1[i][k]=0.;arr1[k][i]=0.;}
        }

        for(i=0;i<n;i++){
                for(k=0;k<n;k++)arr2[i][k]=0.;
        }

        for(j=0;j<=r;j++){
                a=powl(2,r-j);
                WAVE23(arr1,arr2,n/(2*a),-1);
                for(i=0;i<n/a;i++){
                        for(k=0;k<n/a;k++)arr1[i][k]=arr2[i][k];
                }
        }

        for(i=0;i<128;i++){
                for(k=0;k<8;k++){
                        p=16*k;
                        for(j=0;j<16;j++)fprintf(fp3,"%4d",(int)arr1[i][j+p]);fprintf(fp3,"\n");
                }
        }

        fclose(fp1);
        fclose(fp2);
        fclose(fp3);

        for(j=0;j<n+1;j++)delete []arr1[j];
        delete []arr1;
        for(j=0;j<n+1;j++)delete []arr2[j];
        delete []arr2;
        delete []s2;
        delete []s1;

}
```

Appendix(4.7) (Applying a Two-dimensional Wavelet to Compress an Image)

```
// twave24.cpp
// Author: M. D. Choudhry
//
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include "img.h"
#define n 128

void main()
{
```

```
int i,j,k,a,p,q,r=1;
FILE *fp1,*fp2,*fp3;
float **arr1,**arr2,*s1,*s2;

s1=new float [n+1];
s2=new float [n+1];
arr1=new float *[n+1];
for(j=0;j<n+1;j++)arr1[j]=new float [n+1];
arr2=new float *[n+1];
for(j=0;j<n+1;j++)arr2[j]=new float [n+1];

if((fp1=fopen("mona1.dat", "rb"))==NULL) {
        printf("Cannot open input file.\n");exit(1);}
if((fp2=fopen("twave24c.dat", "wb"))==NULL) {
        printf("Cannot open output file.\n");exit(1);}
if((fp3=fopen("twave24r.dat", "wb"))==NULL) {
        printf("Cannot open output file.\n");exit(1);}

for(i=0;i<n;i++){
        for(j=0;j<n;j++)fscanf(fp1,"%4f",&arr1[i][j]);
}

for(j=0;j<=r;j++){
        a=powl(2,j);
        WAVE24(arr1,arr2,n/a);
        for(i=0;i<n/(2*a);i++){
                for(k=0;k<n/(2*a);k++)arr1[i][k]=arr2[i][k];
        }
}

q=n/(2*a);

for(i=0;i<n;i++){
        for(k=0;k<8;k++){
                p=16*k;
                for(j=0;j<16;j++)fprintf(fp2,"%4d",(int)arr1[i][j+p]);fprintf(fp2,"\n");
        }
}

for(i=q;i<n;i++){
        for(k=0;k<n;k++){arr1[i][k]=0.;arr1[k][i]=0.;}
}

for(k=0;k<q;k++){
        for(i=0;i<q;i++)s2[i]=arr1[i][k];
        for(j=0;j<=r;j++){
                a=powl(2,r-j);
                WAVE(s2,s1,n/(2*a),-1);
                for(i=0;i<n/a;i++){
                        s2[i]=s1[i];
                }
        }
        for(i=0;i<n;i++)arr1[i][k]=s1[i];
}
```

```
        for(k=0;k<n;k++){
                for(i=0;i<q;i++)s2[i]=arr1[k][i];
                for(j=0;j<=r;j++){
                        a=powl(2,r-j);
                        WAVE(s2,s1,n/(2*a),-1);
                        for(i=0;i<n/a;i++){
                                s2[i]=s1[i];
                        }
                }
                for(i=0;i<n;i++)arr1[k][i]=s1[i];
        }


        for(i=0;i<128;i++){
                for(k=0;k<8;k++){
                        p=16*k;
                        for(j=0;j<16;j++)fprintf(fp3,"%4d",(int)arr1[i][j+p]);fprintf(fp3,"\n");
                }
        }
        fclose(fp1);
        fclose(fp2);
        fclose(fp3);
        for(j=0;j<n+1;j++)delete []arr1[j];
        delete []arr1;
        for(j=0;j<n+1;j++)delete []arr2[j];
        delete []arr2;
        delete []s2;
        delete []s1;
}
```

Appendix(4.8) (A two-dimensional Data Compressing Wavelet Transform and its Inverse)

```
// twave25.cpp
// Author: M. D. Choudhry
//
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include "img.h"
#define n 128

void main()
{
        int i,j,k,a,p,q,r=0;
        FILE *fp1,*fp2,*fp3;
        float **arr1,**arr2,*s1,*s2;

        s1=new float [n+1];
        s2=new float [n+1];
        arr1=new float *[n+1];
```

```
for(j=0;j<n+1;j++)arr1[j]=new float [n+1];
arr2=new float *[n+1];
for(j=0;j<n+1;j++)arr2[j]=new float [n+1];

if((fp1=fopen("mona1.dat", "rb"))==NULL) {
        printf("Cannot open input file.\n");exit(1);}
if((fp2=fopen("twave25c.dat", "wb"))==NULL) {
        printf("Cannot open output file.\n");exit(1);}
if((fp3=fopen("twave25r.dat", "wb"))==NULL) {
        printf("Cannot open output file.\n");exit(1);}

for(i=0;i<n;i++){
        for(j=0;j<n;j++)fscanf(fp1,"%4f",&arr1[i][j]);
}
for(j=0;j<=r;j++){
        a=powl(2,j);
        WAVE25(arr1,arr2,n/a,1);
        for(i=0;i<n/(2*a);i++){
                for(k=0;k<n/(2*a);k++)arr1[i][k]=arr2[i][k];
        }
}
q=n/(2*a);
for(i=0;i<n;i++){
        for(k=0;k<8;k++){
                p=16*k;
                for(j=0;j<16;j++)fprintf(fp2,"%4d",(int)arr1[i][j+p]);fprintf(fp2,"\n");
        }
}
for(i=q;i<n;i++){
        for(k=0;k<n;k++){arr1[i][k]=0.;arr1[k][i]=0.;}
}
for(i=0;i<n;i++){
        for(k=0;k<n;k++)arr2[i][k]=0.;
}
for(j=0;j<=r;j++){
        a=powl(2,r-j);
        WAVE25(arr1,arr2,n/(2*a),-1);
        for(i=0;i<n/a;i++){
                for(k=0;k<n/a;k++)arr1[i][k]=arr2[i][k];
        }
}
for(i=0;i<128;i++){
        for(k=0;k<8;k++){
                p=16*k;
                for(j=0;j<16;j++)fprintf(fp3,"%4d",(int)arr1[i][j+p]);fprintf(fp3,"\n");
        }
}
fclose(fp1); fclose(fp2); fclose(fp3);
for(j=0;j<n+1;j++)delete []arr1[j];
delete []arr1;
for(j=0;j<n+1;j++)delete []arr2[j];
delete []arr2;
delete []s1; delete []s2;
}
```

Appendix 5

Appendix(5.1)

```
// smooth11.cpp is 2D smoothing wavelet transform for images.
// Author: M. D. Choudhry
//
#include <stdio.h>
#include <stdlib.h>

main()
{
        int i,j,r;
        int **a,**b;
        FILE *fp1,*fp2;

        a=new int *[129];
        for(j=0;j<129;j++)a[j]=new int [129];
        b=new int *[129];
        for(j=0;j<129;j++)b[j]=new int [129];

        if((fp1=fopen("mona1.dat", "r+"))==NULL) {
                printf("Cannot open input file.\n");exit(1);}
        if((fp2=fopen("smooth11.dat", "w+"))==NULL) {
                printf("Cannot open output file.\n");exit(1);}

        for(i=0;i<128;i++){ // Read a file to the array b[][].
                for(j=0;j<128;j++)fscanf(fp1,"%4d",&b[i][j]);
        }

        for(r=1;r<=5;r++){
                // process first row of the image
                a[0][0]=(2*b[0][0]+b[0][1])/3;
                for(j=1;j<=126;j++)a[0][j]=(b[0][j-1]+2*b[0][j]+b[0][j+1])/4;
                a[0][127]=(b[0][126]+2*b[0][127])/3;
                // Process all rows except the first and the last row.
                for(i=1;i<=126;i++){
                a[i][0]=(b[i-1][0]+2*b[i][0]+b[i+1][0]+b[i-1][1]+b[i][1]+b[i+1][1])/7;
                for(j=1;j<=126;j++)a[i][j]=(b[i-1][j-1]+b[i][j-1]+b[i+1][j-1]+b[i-1][j]+2*b[i][j]+
                                        b[i+1][j]+b[i-1][j+1]+b[i][j+1]+b[i+1][j+1])/10;
                                a[i][127]=(b[i-1][126]+b[i][126]+b[i+1][126]+
                                        b[i-1][127]+2*b[i][127]+b[i+1][127])/7;
                }
                // Process the last row
                a[127][0]=(2*b[127][0]+b[127][1])/3;
                for(j=1;j<=126;j++)a[127][j]=(b[127][j-1]+2*b[127][j]+b[127][j+1])/4;
                a[127][127]=(b[127][126]+2*b[127][127])/4;

                for(i=0;i<128;i++){
                    for(j=0;j<128;j++)b[i][j]=a[i][j];
                }
        }
```

```
        for(i=0;i<128;i++){
                for(j=0;j<128;j++)fprintf(fp2,"%4d",a[i][j]);fprintf(fp2,"\n");
        }
        fclose(fp1); fclose(fp2);
        for(j=0;j<129;j++)delete[] a[j];
        delete[] a;
        for(j=0;j<129;j++)delete[] b[j];
        delete[] b;
        return 0;
}
```

Appendix(5.2)

```
// tfpsigdz.cpp
// To find peaks and bottoms of a signal and set the other data elements
// equals zero; Input signal will go in f[] and result comes in pf[].
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>
#include "nrzozi.h"
#define n 128

void main(void)
{
        int i,j;
        int Xstep,Ystep,Xpos;
        float *f,*pf;

        f=(float *)calloc(n+1, sizeof(float));
        pf=(float *)calloc(n+1, sizeof(float));

        Xpos=1;
        Xstep=2;
        Ystep=-16;
        INITIAL();SCRBLACK();

        COSINE(f,n,8);
        XPLOT(f,n,Xstep,Ystep,Xpos,50,GREEN);
        FPSDZ(f,pf,n);
        XPLOT(pf,n,Xstep,Ystep,Xpos,100,LIGHTGRAY);

        VFCOS(f,n);
        XPLOT(f,n,Xstep,Ystep,Xpos,150,GREEN);
        FPSDZ(f,pf,n);
        XPLOT(pf,n,Xstep,Ystep,Xpos,200,LIGHTGRAY);
        getch();
        closegraph();
        free (f);free (pf);
}
```

Appendix(5.3)

```
// tfpsdzr.cpp
// To demonstrate the reconstruct of data between adjacent peaks
// by using the new wavelet transform introduced in Section(8.7).
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>

#include "nrzozi.h"

#define n 128

void main(void)
{
        int Xstep,Ystep,Xpos;
        float *f,*pf,*fo;

        f=(float *)calloc(n+1, sizeof(float));
        pf=(float *)calloc(n+1, sizeof(float));
        fo=(float *)calloc(n+1, sizeof(float));

        Xpos=1;
        Xstep=2;
        Ystep=-16;

        INITIAL();
        SCRBLACK();

        COSINE(f,n,8);
        XPLOT(f,n,Xstep,Ystep,Xpos,50,GREEN);

        FPSDZ(f,pf,n);
        XPLOT(pf,n,Xstep,Ystep,Xpos,100,MAGENTA);

        FPSDZR(pf,fo,n);
        XPLOT(fo,n,Xstep,Ystep,Xpos,150,LIGHTGRAY);

        NRZO(fo,pf,n,16.0);
        XPLOT(pf,n,Xstep,Ystep/2,Xpos,200,LIGHTGRAY);

        getch();

        closegraph();

        free (f);
        free (pf);
        free (fo);

}
```

Appendix(5.4)

```
// tddgauss.cpp
// To demonstrate a different way of developing a Gaussian.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

#include "nrzozi.h"

#define n 128

void main(void)
{
        int i,j,k,nn,a,m;

        int width,Xstep,Ystep,Xpos,tstep;

        float *f,*fo,*nf,sum;

        f=(float *)calloc(n+1, sizeof(float));
        fo=(float *)calloc(2*n+1, sizeof(float));
        nf=(float *)calloc(n+1, sizeof(float));

        Xpos=1;
        Xstep=8;
        Ystep=-40;

        INITIAL();
        SCRWHITE();

         m=8;
        NTANGLE(f,m);
        NR(f,fo,n,m);

        XPLOT(f,m+1,Xstep,Ystep,Xpos,70,BLUE);
        XPLOT(fo,m-1,Xstep,Ystep,Xpos,140,BLUE);
        XPLOT(fo,2*m-1,Xstep,Ystep,Xpos,210,BLUE);

        getch();

        closegraph();

        free (f);
        free (fo);
        free (nf);

}
```

Appendix(5.5)

```
// trpbgauss.cpp
// to Test the Reconstruction of data between adjacent Peaks and Bottoms by
// using GAUSSian wavelet on the style of new WT introduced in Section(8.7).
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include "nrzozi.h"

#define n 128

void main(void)
{
      int Xstep,Ystep,Xpos;

      float *f,*pf,*fo;

      f=(float *)calloc(n+1, sizeof(float));
      pf=(float *)calloc(n+1, sizeof(float));
      fo=(float *)calloc(n+1, sizeof(float));

      Xpos=1;
      Xstep=2;
      Ystep=-16;

      INITIAL();
      SCRWHITE();

      // input COSINE signal
      COSINE(f,n,4);XPLOT(f,n,Xstep,Ystep,Xpos,60,BLUE);

      // Find Peaks and a Signal and set other Data equals Zero
      FPSDZ(f,pf,n); XPLOT(pf,n,Xstep,Ystep,Xpos,120,BLUE);

      // Reconstruct data between Peaks and Bottoms using LoGiSTiC function
      RPBLGSTC(pf,fo,n); XPLOT(fo,n,Xstep,Ystep,Xpos,180,BLUE);

      // input Variant Frequency COSine signal
      VFCOS(f,n); XPLOT(f,n,Xstep,Ystep,Xpos,240,BLUE);

      // Find Peaks and a Signal and set other Data equals Zero
      FPSDZ(f,pf,n); XPLOT(pf,n,Xstep,Ystep,Xpos,300,BLUE);

      // Reconstruct data between Peaks and Bottoms using LoGiSTiC function
      RPBLGSTC(pf,fo,n); XPLOT(fo,n,Xstep,Ystep,Xpos,360,BLUE);

      getch();
      closegraph();
      free (f);free (pf); free (fo);
}
```

Appendix(5.6)

```
// tfpidz.cpp
// To find peaks and bottoms of an image and
// set other data elements equals zero.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>

#include "nrzozi.h"

# define n 128

main()
{
     int i,r,Xstep,Xpos;
     int *fi,*fo;
     float Ystep;
     FILE *fp1,*fp2;

     fi=(int *)calloc(n+1, sizeof(int));
     fo=(int *)calloc(n+1, sizeof(int));

     if((fp1=fopen("mssmthd.dat", "rb"))==NULL) {
                printf("Cannot open input file.\n");exit(1);}
     if((fp2=fopen("tfpidzo.dat", "wb"))==NULL) {
                printf("Cannot open output file.\n");exit(1);}

     Xpos=1;
     Xstep=2;
     Ystep=-.2;
     INITIAL();
     SCRBLACK();

     for(r=0;r<n;r++){
          for(i=0;i<n;i++)fscanf(fp1,"%4d",&fi[i]);
          XPLOTI(fi,n,Xstep,Ystep,Xpos,50+r*2,LIGHTGRAY);
          FPIDZ(fi,fo,n);
          XPLOTI(fo,n,Xstep,Ystep,256,50+r*2,LIGHTGRAY);
          for(i=0;i<n;i++){
            fprintf(fp2,"%4d",fo[i]);
          }
          fprintf(fp2,"\n");
     }
     getch();
     closegraph();
     fclose(fp1); fclose(fp2);
     free (fi); free (fo);
     return 0;
}
```

Appendix(5.7)

```
// tfpidzr.cpp
// Reconstructs the image by using the style of wavelet transform.
// introduced in Section(8.7).
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>

#include "nrzozi.h"
# define n 128

main()
{
     int i,r,Xstep,Xpos;
     int *fi,*fo;
     float Ystep;
     FILE *fp1,*fp2;

     fi=(int *)calloc(n+1, sizeof(int));
     fo=(int *)calloc(n+1, sizeof(int));

     if((fp1=fopen("tfpidzo.dat", "rb"))==NULL) {
                printf("Cannot open input file.\n");exit(1);}
     if((fp2=fopen("tfpidzor.dat", "wb"))==NULL) {
                printf("Cannot open output file.\n");exit(1);}
     Xpos=1;
     Xstep=2;
     Ystep=-.2;

     INITIAL();
     SCRBLACK();

     for(r=0;r<n;r++){
          for(i=0;i<n;i++)fscanf(fp1,"%4d",&fi[i]);
          XPLOTI(fi,n,Xstep,Ystep,Xpos,50+r*2,LIGHTGRAY);
          FPIDZR(fi,fo,n);
          XPLOTI(fo,n,Xstep,Ystep,256,50+r*2,LIGHTGRAY);
          for(i=0;i<n;i++){
             fprintf(fp2,"%4d",fo[i]);
          }
          fprintf(fp2,"\n");
     }
      getch();
      closegraph();
      fclose(fp1);fclose(fp2);
      free (fi);free (fo);
      return 0;
}
```

Appendix(5.8)

```
// twkount.cpp
// To count the number of cusps and troughs of each row and
// and out a two digit binary number to show  the status of the
// the first and the last wave as a cusp or/and as a troughs;
// for example "01" shows that the first wave is a trough and the
// and the last is a cusp.
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>
#include "nrzozi.h"
# define n 128

main()
{
     int i,j,k,r,diff1,diff2,kount;
     int *fi,*fo;
     FILE *fp1,*fp2;

     fi=(int *)calloc(n+1, sizeof(int));
     fo=(int *)calloc(n+1, sizeof(int));

     if((fp1=fopen("tfpidzo.dat", "rb"))==NULL) {
               printf("Cannot open input file.\n");exit(1);}
     if((fp2=fopen("wkount.dat", "wb"))==NULL) {
               printf("Cannot open output file.\n");exit(1);}

     for(j=0;j<n;j++){
          for(i=0;i<n;i++)fscanf(fp1,"%4d",&fi[i]);
          kount=0;
          for(k=1;k<n;k++){
             r=k;
             while(fi[k]==0)k=k+1;
             diff1=fi[k]-fi[r-1];
             if(diff1>0)fo[kount]=1;
             else fo[kount]=0;
             kount=kount+1;
          }
          fprintf(fp2,"%1d",fo[0]);
          fprintf(fp2,"%1d",fo[kount-1]);
          fprintf(fp2,"%4d",kount/2);fprintf(fp2,"\n");
     }

     fclose(fp1);
     fclose(fp2);
     free (fi);
     free (fo);
     return 0;
}
```

Appendix(5.9)

```
// topdpp1.cpp
// To test the logic of Optimal Peaks Detecting Parallel Processor (10.2.4).
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <stdio.h>
#include "nrzozi.h"
#define m 128
#define ssize 8

void main(void)
{
      int i,k,L,n,np,r,maxin,net,Xstep,Ystep,Xpos,*out;
      float *f,*fo,d,d1,d2;

      f=(float *)calloc(m+1, sizeof(float));
      out=(int *)calloc(ssize+2, sizeof(int));
      n=m/2;
      fo=(float *)calloc(n+1, sizeof(float));

      Xpos=1;Xstep=4;Ystep=-8;
      INITIAL();
      SCRBLACK();

      for(np=1;np<=4;np++){
            for(i=0;i<=m;i++)f[i]=0;
            for(i=0;i<=n;i++)fo[i]=0;
            COSINE(f,m,2*np);
            for(i=0;i<=n;i++)f[i]=1.2+f[i];
            XPLOT(f,n,Xstep,Ystep,Xpos,np*70,GREEN);
            for(k=0;k<n/ssize;k++){
                  for(i=0;i<=ssize;i++)out[i]=0;

                  L=1; //layer1
                  for(i=1;i<=ssize-L;i++){
                        d=f[ssize*k+i]-f[ssize*k+i-1];
                        if(d>=0.)out[i-1]=1;
                        else out[i-1]=0;
                  }

                  L=L+1; //layer2
                  for(i=1;i<=ssize-L;i++){
                        net=(i+1)*out[i]+out[i-1];
                        maxin=max((i+1)*out[i],out[i-1]);
                        if(net==1)out[i-1]=1;
                        else if(net>=1)out[i-1]=maxin;
                        else out[i-1]=0;
                  }
```

```
                L=L+1;
                for(r=L;r<=ssize-2;r++){ //layer3-layer(ssize-2)
                    for(i=1;i<=ssize-r;i++){
                        net=out[i]+out[i-1];
                        maxin=max(out[i],out[i-1]);
                        if(net==1)out[i-1]=1;
                        if(net>1)out[i-1]=maxin;
                        else out[i-1]=0;
                    }
                }

                // output layer
                maxin=max(out[1],out[0]);
                if(maxin>=1)fo[maxin+ssize*k]=maxin+ssize*k;

                // Neurons to connect circuit-segments
                for(i=0;i<=2;i++){
                   d=f[ssize*k+ssize+i]-f[ssize*k+ssize+i-1];
                   if(d>=0.)out[i]=1;
                   else out[i]=0;
                }
                if((out[0]+out[1])==2 || (out[1]+out[2])==2) fo[maxin+ssize*k]=0.;
            }
            XPLOT(fo,n+1,Xstep,Ystep/8,Xpos,np*70,LIGHTGRAY);
        }
        getch();
        closegraph();
        free (f);free (fo);free (out);
}
```

Appendix(5.10)

```
// topdpp2.cpp
// To test  the failure of Optimal Peaks Detecting Parallel Processor shown in Figure(10.2.4).
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <stdio.h>
#include "nrzozi.h"
#define m 128
#define ssize 8

void main(void)
{
        int i,k,L,n,np,r,maxin,net,Xstep,Ystep,Xpos,*out;
        float *f,*fo,d,d1,d2;

        f=(float *)calloc(m+1, sizeof(float));
```

```
        out=(int *)calloc(ssize+2, sizeof(int));
        n=m/2;
        fo=(float *)calloc(n+1, sizeof(float));
        Xpos=1;Xstep=4;Ystep=-8;
        INITIAL();SCRBLACK();

        for(np=5;np<=8;np++){
            for(i=0;i<=m;i++)f[i]=0;
            for(i=0;i<=n;i++)fo[i]=0;
            COSINE(f,m,2*np);
            for(i=0;i<=n;i++)f[i]=1.2+f[i];
            XPLOT(f,n,Xstep,Ystep,Xpos,(np-4)*70,GREEN);
            for(k=0;k<n/ssize;k++){
                for(i=0;i<=ssize;i++)out[i]=0;
                L=1; //layer1
                for(i=1;i<=ssize-L;i++){
                    d=f[ssize*k+i]-f[ssize*k+i-1];
                    if(d>=0.)out[i-1]=1;
                    else out[i-1]=0;
                }
                L=L+1; //layer2
                for(i=1;i<=ssize-L;i++){
                    net=(i+1)*out[i]+out[i-1];
                    maxin=max((i+1)*out[i],out[i-1]);
                    if(net==1)out[i-1]=1;
                    else if(net>=1)out[i-1]=maxin;
                    else out[i-1]=0;
                }
                L=L+1;
                for(r=L;r<=ssize-2;r++){ //layer3-layer(ssize-2)
                    for(i=1;i<=ssize-r;i++){
                        net=out[i]+out[i-1];
                        maxin=max(out[i],out[i-1]);
                        if(net==1)out[i-1]=1;
                        if(net>1)out[i-1]=maxin;
                        else out[i-1]=0;
                    }
                }
                //output layer
                maxin=max(out[1],out[0]);
                if(maxin>=1)fo[maxin+ssize*k]=maxin+ssize*k;

                // Neurons to connect circuit-segments
                for(i=0;i<=2;i++){
                   d=f[ssize*k+ssize+i]-f[ssize*k+ssize+i-1];
                   if(d>=0.)out[i]=1;
                   else out[i]=0;
                }
                if((out[0]+out[1])==2 || (out[1]+out[2])==2) fo[maxin+ssize*k]=0.;

            } XPLOT(fo,n+1,Xstep,Ystep/8,Xpos,(np-4)*70,LIGHTGRAY);
        } getch();
        closegraph();free (f);free (fo);free (out);
}
```

Appendix(5.11)

```
// topdpp3.cpp
// To test the logic of  Optimal Peaks Detecting  Parallel Processor  shown in
// Figure(10.2.4) for the cosine signals having the same frequencies for which
// the program topdpp2.cpp failed to produced correct results
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <stdio.h>
#include "nrzozi.h"
#define m 128
#define ssize 4

void main(void)
{
      int i,k,L,n,np,r,maxin,net,Xstep,Ystep,Xpos,*out;
      float *f,*fo,d,d1,d2;

      f=(float *)calloc(m+1, sizeof(float));
      out=(int *)calloc(ssize+2, sizeof(int));
      n=m/2;
      fo=(float *)calloc(n+1, sizeof(float));

      Xpos=1;Xstep=4;Ystep=-8;
      INITIAL();SCRBLACK();

      for(np=5;np<=8;np++){
            for(i=0;i<=m;i++)f[i]=0;
            for(i=0;i<=n;i++)fo[i]=0;
            COSINE(f,m,2*np);
            for(i=0;i<=n;i++)f[i]=1.2+f[i];
            XPLOT(f,n,Xstep,Ystep,Xpos,(np-4)*70,GREEN);
            for(k=0;k<n/ssize;k++){
                  for(i=0;i<=ssize;i++)out[i]=0;

                  L=1; //layer1
                  for(i=1;i<=ssize-L;i++){
                        d=f[ssize*k+i]-f[ssize*k+i-1];
                        if(d>=0.)out[i-1]=1;
                        else out[i-1]=0;
                  }
                  L=L+1; //layer2
                  for(i=1;i<=ssize-L;i++){
                        net=(i+1)*out[i]+out[i-1];
                        maxin=max((i+1)*out[i],out[i-1]);
                        if(net==1)out[i-1]=1;
                        else if(net>=1)out[i-1]=maxin;
                        else out[i-1]=0;
                  }
```

```
                L=L+1;
                for(r=L;r<=ssize-2;r++){ //layer3-layer(ssize-2)
                    for(i=1;i<=ssize-r;i++){
                        net=out[i]+out[i-1];
                        maxin=max(out[i],out[i-1]);
                        if(net==1)out[i-1]=1;
                        if(net>1)out[i-1]=maxin;
                        else out[i-1]=0;
                    }
                }

                //output layer
                maxin=max(out[1],out[0]);
                if(maxin>=1)fo[maxin+ssize*k]=maxin+ssize*k;

                // Neurons to connect circuit-segments
                for(i=0;i<=2;i++){
                   d=f[ssize*k+ssize+i]-f[ssize*k+ssize+i-1];
                   if(d>=0.)out[i]=1;
                   else out[i]=0;
                }
                if((out[0]+out[1])==2 || (out[1]+out[2])==2) fo[maxin+ssize*k]=0.;
            }
            XPLOT(fo,n+1,Xstep,Ystep/8,Xpos,(np-4)*70,LIGHTGRAY);
        }
        getch();
        closegraph();
        free (f);free (fo);free (out);
}
```

Appendix(5.12)

```
// tobdpp.cpp
// To test the logic of Optimal bottom Detecting Parallel Processor suggested in Section(10.2.5).
// Author: M. D. Choudhry
//
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include <stdio.h>
#include "nrzozi.h"
#define m 128
#define ssize 4

void main(void)
{
        int i,k,L,n,np,r,maxin,net,Xstep,Ystep,Xpos,*out;
        float *f,*fo,d,d1,d2;

        f=(float *)calloc(m+1, sizeof(float));
        out=(int *)calloc(ssize+2, sizeof(int));
        n=m/2;
```

```
    fo=(float *)calloc(n+1, sizeof(float));

    Xpos=1;Xstep=4;Ystep=-8;
    INITIAL();SCRBLACK();

    for(np=5;np<=8;np++){
        for(i=0;i<=m;i++)f[i]=0;
        for(i=0;i<=n;i++)fo[i]=0;
        COSINE(f,m,2*np);
        for(i=0;i<=n;i++)f[i]=1.2+f[i];
        XPLOT(f,n,Xstep,Ystep,Xpos,(np-4)*70,GREEN);
        for(k=0;k<n/ssize;k++){
            for(i=0;i<=ssize;i++)out[i]=0;
            L=1; //layer1
            for(i=1;i<=ssize-L;i++){
                d=f[ssize*k+i]-f[ssize*k+i-1];
                if(d<=0.)out[i-1]=1;
                else out[i-1]=0;
            }
            L=L+1; //layer2
            for(i=1;i<=ssize-L;i++){
                net=(i+1)*out[i]+out[i-1];
                maxin=max((i+1)*out[i],out[i-1]);
                if(net==1)out[i-1]=1;
                else if(net>=1)out[i-1]=maxin;
                else out[i-1]=0;
            }
            L=L+1;
            for(r=L;r<=ssize-2;r++){ //layer3-layer(ssize-2)
                for(i=1;i<=ssize-r;i++){
                    net=out[i]+out[i-1];
                    maxin=max(out[i],out[i-1]);
                    if(net==1)out[i-1]=1;
                    if(net>1)out[i-1]=maxin;
                    else out[i-1]=0;
                }
            }
            //output layer
            maxin=max(out[1],out[0]);
            if(maxin>=1)fo[maxin+ssize*k]=-(maxin+ssize*k);

            // Neurons to connect circuit-segments
            for(i=0;i<=2;i++){
               d=f[ssize*k+ssize+i]-f[ssize*k+ssize+i-1];
               if(d<=0.)out[i]=1;
               else out[i]=0;
            }
            if((out[0]+out[1])==2 || (out[1]+out[2])==2) fo[maxin+ssize*k]=0.;
        }
        XPLOT(fo,n+1,Xstep,Ystep/8,Xpos,(np-4)*70,LIGHTGRAY);
    }
    getch();
    closegraph();free (f);free (fo);free (out);
}
```

Subroutines

Subroutine: (wdmrrb.cpp)

```
// wdmrrb.cpp
// Wavelet decomposition with multiresolution relative bandwidth.
// Author: M. D. Choudhry
#include <stdlib.h>
#include <alloc.h>
#include <math.h>
#include "nrzozi.h"

void WDMRRB(float fi[], float fo[],int n,int j, int sgn)
{
        int i,a,k,r,dw,ha,width;
        float *w,*g,*zi;

        g=(float *)calloc(n+1, sizeof(float));
        zi=(float *)calloc(n+1, sizeof(float));
        w=(float *)calloc(n+1, sizeof(float));
        a=pow(2,j);
        dw=n/a;
        if(sgn==1){
              for(r=2;r<=6;r++){
                   width=pow(2,r);
                   GAUSSCR(g,n,width,2*r);
                   for(i=1,k=1;i<=n;k=k+1,i=i+a){
                        zi[k]=g[i]*fi[i];
                        w[k]=w[k]+width*g[i];
                   }
                   for(i=1;i<=dw;i++) fo[i]=fo[i]+width*zi[i];
              }
              for(i=1;i<=dw;i++)fo[i]=fo[i]/w[i];
        }
        if(sgn==-1){
             w[0]=0.;
             for(i=1;i<=a;i++)w[i]=(float)i;
             for(r=0;r<n;r++){
                  for(i=1;i<=a;i++) fo[r*a+i]=(w[a-i+1]*fi[a+r]+w[i-1]*fi[a+r+1])/a;
             }
        } free (w);free (g); free (zi);
}
```

Subroutine: (rnoise.cpp)

```
// rnoise.cpp
// Author: M. D. Choudhry
#include <stdlib.h>

void RNOISE(float si[], float so[], int n, int maxnum)
{
  int i;
  for(i=1;i<=n;i++)so[i]=si[i]+random(maxnum)/2./
}
```

Subroutine: (awczot.cpp)

```
// awczot.cpp
// Automatic wavelet compressing zoom-out
// wavelet transform which reduced noise as well.
// Author: M. D. Choudhry
#include <math.h>
#include <alloc.h>
#include "nrzozi.h"

void AWCZOT(float fi[],float fo[],int n,int m,int j)
{
        int i,k,a,nn,r,hm,mm;
        float *w,*s,sum,lemda;

        s=(float *)calloc(n+2*m+1, sizeof(float));
        w=(float *)calloc(2*m+1, sizeof(float));

        if(j<=-0){ // Noise reduction and Zoom-out
                // Compute energy of the basic wavelet.
                lemda=0.0;
                for(i=1;i<=m/2;i++){
                     lemda=lemda+2*i;
                }
                lemda=pow(lemda,.5);
                hm=m/2;

                for(i=1;i<=hm;i++){
                     w[i]=i/lemda;
                     w[i+hm]=(hm-i+1)/lemda;
                }

                // To perform zero-padding
                a=pow(2,-j);
                mm=m/a;
                for(i=0;i<=mm;i++){ s[i]=0.;s[n+i]=0.; }
                for(i=1;i<=n;i++){
                     s[i+mm/2]=fi[i];
                }

                // Perform the transform
                for(k=0;k<n;k++){
                        for(r=1;r<=a;r++){
                                sum=0.;
                                for(i=1;i<=mm;i++){
                                     sum=sum+w[a*i]*s[i+k];
                                }
                                fo[a*k+r]=sum;
                        }
                }
        }
        free (s);
}
```

Subroutine: (awezit.cpp)

```
// awezit.cpp
// Automatic wavelet expanding zoom-in wavelet transform.
// (The code for vertical energy balancing will be included later)
// Author: M. D. Choudhry
#include <math.h>
#include <alloc.h>
#include "nrzozi.h"

void AWEZIT(float fi[],float fo[],int n,int m,int j)
{
        int i,k,a,nn,mm;
        float *s,*w,sum,lemda;

        if(j>=0){
                s=(float *)calloc(n+2*m+1, sizeof(float));
                a=pow(2,j);
                w=(float *)calloc(a*m+1, sizeof(float));

                // Compute energy of the basic wavelet.
                for(i=1;i<=m/2;i++){
                     lemda=lemda+2*i;
                }
                lemda=pow(lemda,.5);

                // Expand basic wavelet according to required resolution;
                mm=a*m;
                for(i=1;i<=mm/2;i++){
                      w[i]=i/(a*lemda);
                      w[-i+1+mm/2]=(-i+1+mm/2)/(a*lemda);
                }

                // Initialise temp storage;
                for(i=0;i<=mm/2;i++){ s[i]=0.;s[n+i]=0.; }

                // To perform zero-padding
                for(i=1;i<=n;i++){
                      s[i+mm/2]=fi[i];
                }

                // Perform the wavelet transform;
                nn=n/a;
                for(k=0;k<nn;k++){
                     sum=0.;
                     for(i=1;i<=mm;i++){
                           sum=sum+w[i]*s[i+a*k];
                     }
                     fo[k]=sum;
                }
        }
        free (s);
}
```

Subroutine: (nrzo.cpp)

```
// nrzo.cpp
// A noise reducing zoom-out wavelet transform wchich
// uses a wavelet keeping it size the fixed.
// (the code for vertical normalisation of the output
//  will be developed later)
// Author: M. D. Choudhry
#include <math.h>
#include <alloc.h>

#include "nrzozi.h"

void NRZO(float fi[],float fo[],int n,int m,int j)
{
        int i,k,a,nn,r,hm;
        float *s,*w,sum,lemda;

        if(j<=-0){
                s=(float *)calloc(n+2*m+1, sizeof(float));
                w=(float *)calloc(m+1, sizeof(float));

                // Construct a normalised wavelet of width m and hight m/2
                hm=m/2;
                for(i=1;i<=hm;i++){
                        w[i]=i;
                        w[hm+1-i]=hm+1-i;
                        lemda=lemda+2*i;
                }
                lemda=pow(lemda,.5);
                for(i=1;i<=hm;i++)w[i]=w[i]/lemda;

                // For zero-padding
                for(i=0;i<=m;i++){
                        s[i]=0.;
                        s[n+i]=0.;
                }
                for(i=1;i<=n;i++){
                        s[i+hm]=fi[i];
                }
                a=pow(2,-j);
                for(k=0;k<n;k++){
                        for(r=1;r<=a;r++){
                                sum=0.;
                                for(i=1;i<=m;i++){
                                        sum=sum+w[i]*s[i+k];
                                }
                                fo[a*k+r]=sum;
                        }
                }
                 free (s); free (w);
        }
}
```

Subroutine: (nr.cpp)

```
// nr.cpp
// Noise Reducing wavelet transform wchich uses
// the identity wavelet and keeps it size the fixed.
// Author: M. D. Choudhry
//
#include <math.h>
#include <alloc.h>

#include "nrzozi.h"

void NR(float fi[],float fo[],int n,int m)
{
        int i,k,hm;
        float *s,*w,sum,lemda;

        s=(float *)calloc(n+2*m+1, sizeof(float));
        w=(float *)calloc(m+1, sizeof(float));

        // Construct a normalised wavelet of width m and hight m/2
        hm=m/2;
        for(i=1;i<=hm;i++){
                w[i]=(float)i/hm;
                w[hm+i]=(float)(hm+1-i)/hm;
                lemda=lemda+w[i]+w[hm+i];
        }

        //lemda=pow(lemda,.5);
        for(i=0;i<=m;i++)w[i]=w[i]/lemda;

        // For zero-padding
        for(i=0;i<=m;i++){
                s[i]=0.;
                s[n+i]=0.;
        }
        for(i=0;i<n;i++){
                s[i+m-1]=fi[i];
        }

        // Perform the transform
        for(k=0;k<n+2*m;k++){
                sum=0.;
                for(i=0;i<=m;i++){
                   sum=sum+w[i]*s[i+k];
                }
                fo[k]=sum;
        }

        free (s);
        free (w);

}
```

Subroutine: (nrzi.cpp)

```
// nrzi.cpp
// A wavelet transform which reduces noise
// and compresses data simultaneously.
// Author: M. D. Choudhry
//
#include <math.h>
#include <alloc.h>
#include "nrzozi.h"

void NRZI(float fi[],float fo[],int n,int m,int j)
{
        int i,k,a,nn,r,hm;
        float *s,*w,sum,lemda;

        // Noise reduction and Zoom-in
        if(j>=0){
                s=(float *)calloc(n+2*m+1, sizeof(float));
                w=(float *)calloc(m+1, sizeof(float));

                // Construct a normalised wavelet of width m and hight m/2
                hm=m/2;
                for(i=1;i<=hm;i++){
                        w[i]=i;
                        w[hm+1-i]=hm+1-i;
                        lemda=lemda+2*i;
                }
                lemda=pow(lemda,.5);
                for(i=1;i<=hm;i++)w[i]=w[i]/lemda;

                // For zero zero-padding
                for(i=0;i<=m;i++){
                        s[i]=0.;
                        s[n+i]=0.;
                }
                for(i=1;i<=n;i++){
                        s[i+hm]=fi[i];
                }

                // Compute the transform;
                a=pow(2,j);
                nn=n/a;
                for(k=0;k<nn;k++){
                        sum=0.;
                        for(i=1;i<=m;i++){
                                sum=sum+w[i]*s[i+a*k];
                        }
                        fo[k]=sum;
                }
                free (s); free (w);
        }
}
```

Subroutine: (zoomout.cpp)

```
// zoomout.cpp
// A wavelet transform which expands data without smoothing it
// Author: M. D. Choudhry
//
#include <math.h>

void ZOOMOUT(float fi[], float fo[],int n,int j)
{
   int i,k,r,a;

   a=pow(2,j);
   for(r=0;r<n;r++){
      for(i=1;i<=a;i++){
         fo[r*a+i]=((a-i+1)*fi[a+r]+(i-1)*fi[a+r+1])/a;
      }
   }
}
```

Subroutine: (fdbpb.cpp)

```
// fdbpb.cpp
// To find difference between amplitudes of adjacent peaks and bottoms of a signal.
// Author: M. D. Choudhry
//
#include <math.h>
#include "nrzozi.h"

void FDBPB(float fi[], float fo[], int n)
{
        int i,j,k,r;
        float a1,a2;

        FPSDZ(fi,fo,n);

        k=0;
        while(fo[k]==0.)k=k+1;
        a1=fo[k];
        for(i=0;i<=k;i++)fo[i]=abs(a1);
        r=k+1;

        for(i=r;i<n;i++){
              k=0;
              while(fo[i]==0.){i=i+1;k=k+1;}
              a2=fo[i];
              for(j=0;j<=k;j++)fo[i-k+j]=abs(a1-a2);
              a1=a2;
        }

}
```

Subroutine: (fpsdz.cpp)

```
// fpsdz.cpp
// To find peaks and bottoms of a signal and set the other data elements equals zero.
// Author: M. D. Choudhry
//
void FPSDZ(float fi[], float fo[], int n)
{
   int j,r;
   float diff1,diff2;
   for(r=0;r<n;r++)fo[r]=fi[r];
   for(r=1;r<=n;r++){
       diff1=fo[1]-fo[0];
       for(j=1;j<127;j++){
           diff2=fo[j+1]-fo[j];
           if(!((diff1>0 && diff2<=0)||(diff1<0 && diff2>=0)))fo[j]=0;
           diff1=diff2;
       }
   }
}
```

Subroutine: (wave.cpp)

```
// wave.cpp
// Author: M. D. Choudhry
//
#include <math.h>

void WAVE(float f[],float s[],int n,int j,int sgn)
{
        int i,k;
        float m1=.5,m2=4.0;
        float d[4]={.1,.4,.4,.1};

        // Compute Wavelet Transform
        if(sgn==1){
                s[0]=m1*2*(d[2]*f[0]+d[3]*f[1]);
                k=0;
                for(i=1;i<n/2-1;i++){
                        s[i]=m1*(d[0]*f[k]+d[1]*f[k+1]+d[2]*f[k+2]+d[3]*f[k+3]);
                        k=k+2;
                }
                s[n/2-1]=m1*2*(d[0]*f[n-2]+d[1]*f[n-1]);
        }
        // Compute Inverse Wavelet Transform
        if(sgn==-1){
                s[0]=m2*(d[2]*f[0]+d[0]*f[1]);
                s[1]=m2*(d[3]*f[0]+d[1]*f[1]);
                for(i=0;i<=n-2;i++){
                        s[2*i+2]=m2*(d[2]*f[i]+d[0]*f[i+1]);
                        s[2*i+3]=m2*(d[3]*f[i]+d[1]*f[i+1]);
                }
        }
}
```

Subroutine: (wave23.cpp)

```
// wave23.cpp
// Author: M. D. Choudhry
//
#include <stdio.h>
#include <conio.h>
#include <math.h>

void WAVE23(float *f[],float *s[],int n,int sgn)
{
        int i,j,k,l,p,q;
        float m1=.4,m2=.8,sum=0.;
        float *d;

        d=new float [6];

        d[0]=.1;
        d[1]=.2;
        d[2]=.5;
        d[3]=.2;
        d[0]=.1;

        if(sgn==1){
                // Processing for first row
                s[0][0]=2.7*m1*(f[0][0]*.5+f[0][1]*.2+f[0][2]*.1+
                                                        f[1][0]*.2+
                                                        f[2][0]*.1);
                k=0;
                for(j=1;j<=n/2-2;j++){
                   sum=0.;

sum=sum+(f[0][k]*.1+f[0][k+1]*.2+f[0][k+2]*.5+f[0][k+3]*.2+f[0][k+4]*.1+
                                        f[1][k+2]*.2+
                                        f[2][k+2]*.1);
                  s[0][j]=2*m1*sum;
                  k=k+2;
                }
                s[0][n/2-1]=2.7*m1*(f[0][n-3]*.1+f[0][n-2]*.2+f[0][n-1]*.5+
                                                        f[1][n-1]*.2+
                                                        f[2][n-1]*.1);

                // Processing for all rows execpt the first and the last one
                l=0;
                for(i=1;i<=n/2-2;i++){
                   s[i][0]=1.5*m1*(f[l+0][0]*.1+
                                        f[l+1][0]*.2+
                                        f[l+2][0]*.5+f[l+2][1]*.2+f[l+2][2]*.1+
                                        f[l+3][0]*.2+
                                        f[l+4][0]*.1);


                   k=0;
```

```
            for(j=1;j<=n/2-2;j++){
                sum=0.;
                sum=sum+(       f[l+0][k+2]*.1+
                                f[l+1][k+2]*.2+
                f[l][k]*.1+f[l+2][k+1]*.2+f[l+2][k+2]*.5+f[l+2][k+3]*.2+f[l+2][k+4]*.1+
                                f[l+3][k+2]*.2+
                                f[l+4][k+2]*.1);

                s[i][j]=1.5*m1*sum;
                k=k+2;
            }
            s[i][n/2-1]=2.2*m1*(            f[l+0][n-1]*.1+
                                            f[l+1][n-1]*.2+
                    f[l+2][n-3]*.1+f[l+2][n-2]*.2+f[l+2][n-1]*.5+
                                            f[l+3][n-1]*.2+
                                            f[l+4][n-1]*.1);
            l=l+2;
        }
        // Processing the last row
        s[n/2-1][0]=2.7*m1*(f[n/2-3][0]*.1+
                        f[n/2-2][0]*.2+
                        f[n/2-1][0]*.5+f[n/2-1][1]*.2+f[n/2-1][2]*.1);

        k=0;
        for(j=1;j<=n/2-2;j++){
            sum=0.;
            sum=sum+(               f[n/2-3][k+2]*.1+
                                    f[n/2-2][k+2]*.2+
            f[n/2-1][k]*.1+f[n/2-1][k+1]*.2+f[n/2-1][k+2]*.5+f[n/2-1][k+3]*.2+f[n/2-1][k+4]*.1);
            s[n/2-1][j]=2*m1*sum;
            k=k+2;
        }
        s[n/2-1][n/2-1]=2.7*m1*(        [n/2-3][n-1]*.1+
                                        f[n/2-2][n-1]*.2+
        f[n/2-1][n-3]*.1+f[n/2-1][n-2]*.2+f[n/2-1][n-1]*.5);
    }
    if(sgn==-1){ // Process all rows
            for(i=0;i<n;i++){
                    s[2*i][0]=2*m2*(f[i][0]*.5+f[i+1][0]*.1);
                    s[2*i+1][0]=2*m2*(f[i][0]*.1+f[i+1][0]*.5);
                    for(j=1;j<n;j++){
                        s[2*i][2*j-1]=m2*(f[i][j-1]*.5+f[i+1][j]*.1+f[i+1][j-1]*.1+f[i+1][j]*.5);
                        s[2*i][2*j]=m2*(f[i][j]*.1+f[i][j+1]*.5+f[i+1][j]*.5+f[i+1][j+1]*.1);
                        s[2*i+1][2*j-1]=m2*(f[i][j-1]*.1+f[i][j]*.5+f[i+1][j-1]*.5+f[i+1][j]*.1);
                        s[2*i+1][2*j]=m2*(f[i][j]*.5+f[i][j+1]*.1+f[i+1][j]*.1+f[i+1][j+1]*.5);
                    }
                    s[2*i][2*n-2]=2*m2*(f[i][n-2]*.5+f[i+1][n-1]*.1);
                    s[2*i][2*n-1]=2*m2*(f[i][n-2]*.5+f[i+1][n-1]*.1);
                    s[2*i+1][2*n-2]=2*m2*(f[i][n-2]*.1+f[i+1][n-1]*.5);
                    s[2*i+1][2*n-1]=2*m2*(f[i][n-2]*.1+f[i+1][n-1]*.5);
            }
    }
    delete []d;
}
```

Subroutine: (wave24.cpp)

```
// wave24.cpp
// Author: M. D. Choudhry
#include <stdio.h>
#include <conio.h>
#include <math.h>

void WAVE24(float *f[],float *s[],int n)
{
 int i,j,k,l,p,q;
 float m1=.35,m2=.2,sum=0.;
 float **d;

 d=new float *[n+1];
 for(j=0;j<5;j++)d[j]=new float [5];

 for(j=0;j<4;j++)d[0][j]=d[3][j]=.1;
 d[1][0]=d[1][3]=d[2][0]=d[2][3]=.1;
 d[1][1]=d[1][2]=d[2][1]=d[2][2]=.4;

 // Processing for first row
 s[0][0]=4*m1*(f[0][0]*d[2][2]+f[0][1]*d[2][3]+f[1][0]*d[3][2]+f[1][1]*d[3][3]);
 k=0;
 for(j=1;j<=n/2-2;j++){
          sum=0.;
          for(p=0;p<=1;p++){
                   for(q=0;q<4;q++)sum=sum+f[p][q+k]*d[p+2][q];
          }
          s[0][j]=2*m1*sum;
          k=k+2;
 }
 s[0][n/2-1]=4*m1*(f[0][n-2]*d[2][0]+f[0][n-1]*d[2][1]+f[1][n-2]*d[3][0]+f[1][n-1]*d[3][1]);

 // Processing for all rows execpt the first and the last one
 l=0;
 for(i=1;i<=n/2-2;i++){
          s[i][0]=2*m1*(f[l][0]*d[0][2]+f[l][1]*d[0][3]+f[l+1][0]*d[1][2]+f[l+1][1]*d[1][3]+
                    f[1+2][0]*d[2][2]+f[l+2][1]*d[2][3]+f[l+3][0]*d[3][2]+f[l+3][1]*d[3][3]);
          k=0;
          for(j=1;j<=n/2-2;j++){
                   sum=0.;
                   for(p=0;p<4;p++){
                            for(q=0;q<4;q++)sum=sum+f[p+l][q+k]*d[p][q];
                   }
                   s[i][j]=m1*sum;
                   k=k+2;
          }
          s[i][n/2-1]=2*m1*(f[l][n-2]*d[0][0]+f[l][n-1]*d[0][1]+f[l+1][n-2]*d[1][0]+
                             f[l+1][n-1]*d[1][1]+f[l+2][n-2]*d[2][0]+f[l+2][n-1]*d[2][1]+
                              f[l+3][n-2]*d[3][0]+f[l+3][n-1]*d[3][1]);
          l=l+2;
 }
```

```
    // Processing last row
    s[n/2-1][0]=4*m1*(f[n/2-2][0]*d[0][2]+f[n/2-2][1]*d[0][3]+f[n/2-1][0]*d[1][2]+
    f[n/2-1][1]*d[1][3]);
    k=0;
    for(j=1;j<=n/2-2;j++){
        sum=0.;
        for(p=0;p<=1;p++){
                for(q=0;q<4;q++)sum=sum+f[p+n/2-2][q+k]*d[p][q];
        }
        s[n/2-1][j]=2*m1*sum;
        k=k+2;
    }
    s[n/2-1][n/2-1]=4*m1*(f[n-2][n-2]*d[0][0]+f[n-2][n-1]*d[0][1]+f[n-1][n-2]*d[1][0]+
                                    f[n-1][n-1]*d[1][1]);
    for(j=0;j<5;j++)delete []d[j];
    delete []d;
}
```

Subroutine: (wave25.cpp)

```
// wave25.cpp
// Author: M. D. Choudhry
#include <stdio.h>
#include <conio.h>
#include <math.h>

void WAVE25(float *f[],float *s[],int n,int sgn)
{
   int i,j,k,l,p,q;
   float **d,m1=.35,m2=1.,sum=0.;
   d=new float *[n+1];

   for(j=0;j<5;j++)d[j]=new float [5];
   for(j=0;j<4;j++)d[0][j]=d[3][j]=.1;
   d[1][0]=d[1][3]=d[2][0]=d[2][3]=.1;
   d[1][1]=d[1][2]=d[2][1]=d[2][2]=.4;

   if(sgn==1){
      // Processing for first row
      s[0][0]=4*m1*(f[0][0]*d[2][2]+f[0][1]*d[2][3]+f[1][0]*d[3][2]+f[1][1]*d[3][3]);
      k=0;
      for(j=1;j<=n/2-2;j++){
         sum=0.;
         for(p=0;p<=1;p++){
           for(q=0;q<4;q++)sum=sum+f[p][q+k]*d[p+2][q];
         }
         s[0][j]=2*m1*sum;
         k=k+2;
      }
      s[0][n/2-1]=4*m1*(f[0][n-2]*d[2][0]+f[0][n-1]*d[2][1]+f[1][n-2]*d[3][0]+
                  f[1][n-1]*d[3][1]);
```

```
            // Processing for all rows execpt the first and the last one
            l=0;
            for(i=1;i<=n/2-2;i++){
                s[i][0]=2*m1*(f[l][0]*d[0][2]+f[l][1]*d[0][3]+f[l+1][0]*d[1][2]+
                f[l+1][1]*d[1][3]+f[1+2][0]*d[2][2]+f[l+2][1]*d[2][3]+
                f[l+3][0]*d[3][2]+f[l+3][1]*d[3][3]);
                k=0;
                for(j=1;j<=n/2-2;j++){
                    sum=0.;
                    for(p=0;p<4;p++){
                      for(q=0;q<4;q++)sum=sum+f[p+l][q+k]*d[p][q];
                    }
                    s[i][j]=m1*sum;
                    k=k+2;
                }
                s[i][n/2-1]=2*m1*(f[l][n-2]*d[0][0]+f[l][n-1]*d[0][1]+f[l+1][n-2]*d[1][0]+
                            f[l+1][n-1]*d[1][1]+f[l+2][n-2]*d[2][0]+f[l+2][n-1]*d[2][1]+
                            f[l+3][n-2]*d[3][0]+f[l+3][n-1]*d[3][1]);
                l=l+2;
            }
            // Processing last row
            s[n/2-1][0]=4*m1*(f[n/2-2][0]*d[0][2]+f[n/2-2][1]*d[0][3]+f[n/2-1][0]*d[1][2]+
                        f[n/2-1][1]*d[1][3]);
            k=0;
            for(j=1;j<=n/2-2;j++){
                sum=0.;
                for(p=0;p<=1;p++){
                    for(q=0;q<4;q++)sum=sum+f[p+n/2-2][q+k]*d[p][q];
                }
                s[n/2-1][j]=2*m1*sum;
                k=k+2;
            }
            s[n/2-1][n/2-1]=4*m1*(f[n-2][n-2]*d[0][0]+f[n-2][n-1]*d[0][1]+
                            f[n-1][n-2]*d[1][0]+f[n-1][n-1]*d[1][1]);
    }
    if(sgn==-1){ // Process all rows
            for(i=0;i<n;i++){
                    s[2*i][0]=2*m2*(f[i][0]*.4+f[i+1][0]*.1);
                    s[2*i+1][0]=2*m2*(f[i][0]*.1+f[i+1][0]*.4);
                    for(j=1;j<n;j++){
                        s[2*i][2*j-1]=m2*(f[i][j-1]*.4+f[i][j]*.1+f[i+1][j-1]*.1+f[i+1][j]*.4);
                        s[2*i][2*j]=m2*(f[i][j]*.1+f[i][j+1]*.4+f[i+1][j]*.4+f[i+1][j+1]*.1);
                        s[2*i+1][2*j-1]=m2*(f[i][j-1]*.1+f[i][j]*.4+ f[i+1][j-1]*.4+f[i+1][j]*.1);
                        s[2*i+1][2*j]=m2*(f[i][j]*.4+f[i][j+1]*.1+ f[i+1][j]*.1+f[i+1][j+1]*.4);
                    }
                    s[2*i][2*n-2]=2*m2*(f[i][n-2]*.4+f[i+1][n-1]*.1);
                    s[2*i][2*n-1]=2*m2*(f[i][n-2]*.4+f[i+1][n-1]*.1);
                    s[2*i+1][2*n-2]=2*m2*(f[i][n-2]*.1+f[i+1][n-1]*.4);
                    s[2*i+1][2*n-1]=2*m2*(f[i][n-2]*.1+f[i+1][n-1]*.4);
            }
    }
    for(j=0;j<5;j++)delete []d[j];
    delete []d;
}
```

Subroutine: (fpsdz.cpp)

```
// fpsdz.cpp
// To find peaks and bottoms of a signal and set the other data elementsnts
// equals zero; input signal will be in fi[] and out result in fo[].
// Author: M. D. Choudhry
//
void FPSDZ(float fi[], float fo[], int n)
{
   int j,r;

   float diff1,diff2;

   // make zero the elements except peaks
   for(r=0;r<n;r++)fo[r]=fi[r];
   for(r=1;r<=n;r++){
       diff1=fo[1]-fo[0];
       for(j=1;j<127;j++){
           diff2=fo[j+1]-fo[j];
           if(!((diff1>0 && diff2<=0)||(diff1<0 && diff2>=0)))fo[j]=0;
           diff1=diff2;
       }
   }

}
```

Subroutine: (fpsdzr.cpp)

```
// fpsdzr.cpp
// Subroutine to reconstruct signal data between peaks by using the
// style of zoom-out wavelet transform introduced in Section(8.7).
// Author: M. D. Choudhry
//
void FPSDZR(float fi[], float fo[], int n)
    {
        int i,k,pk,m;

        for(k=1;k<=n;k++){
            pk=k;
            m=1;
            if(fi[k]==0.){
               while(fi[k]==0){k=k+1;m=m+1;}
               for(i=1;i<=m;i++){
                  fo[pk+i-2]=((m+1-i)*fi[pk-1]+(i-1)*fi[k])/m;
               }
            }
        }
    }
```

Subroutine: (rpblgstc.cpp)

```
// rpblgstc.cpp
// Reconstruct data between Peaks and Bottoms by using the LoGiSTiC function.
// Author: M. D. Choudhry
//
# include <alloc.h>
# include "nrzozi.h"

void RPBLGSTC(float fi[], float fo[], int n)
{
   int i,k,pk,m;
   float *w;

   w=(float *)calloc(n+1, sizeof(float));

   for(k=1;k<=n;k++){
       pk=k;
       m=1;
       if(fi[k]==0.){
          while(fi[k]==0){k=k+1;m=m+1;}
          LOGISTIC(w,2*m);
          for(i=1;i<=m;i++){
             fo[pk+i-2]=w[m+1-i]*fi[pk-1]+w[i-1]*fi[k];
          }
       }
   }

   free (w);

}
```

Subroutine: (fpidz.cpp)

```
// fpidz.cpp
// To find peaks and bottoms of an image and set the other data elements equals zero
// Author: M. D. Choudhry
//
void FPIDZ(int fi[], int fo[], int n)
{
   int i,j,k,r,diff1,diff2;

   for(i=0;i<n;i++)fo[i]=fi[i];
   for(i=0;i<n;i++){ // Layer1
       diff1=fo[1]-fo[0];
       for(j=1;j<127;j++){
          diff2=fo[j+1]-fo[j];
          if(!((diff1>0 && diff2<=0)||(diff1<0 && diff2>=0)))fo[j]=0;
          diff1=diff2;
       }
   }
```

```
   for(i=0;i<128;i++){  // Layer2
         k=1;
         while(fo[k]==0)k=k+1;
            diff1=fo[k]-fo[0];
            for(j=k;j<=126;j++){
               r=j;
               while(fo[j+1]==0)j=j+1;
               diff2=fo[j+1]-fo[r];
               if((diff1>=0 && diff2>=0)||(diff1<=0 && diff2<=0))fo[r]=0;
               diff1=diff2;
            }
      }
}
```

Subroutine: (fpidzr.cpp)

```
// fpidzr.cpp
//  to reconstruct data between peaks by using zoom-out transform introduced in Section(8.6).
// Author: M. D. Choudhry
//
void FPIDZR(int fi[], int fo[], int n)
{
   int i,k,pk,m;

   for(k=1;k<=n;k++){
      pk=k;
      m=1;
      if(fi[k]==0.){
         while(fi[k]==0){k=k+1;m=m+1;}
         for(i=1;i<=m;i++){
            fo[pk+i-2]=((m+1-i)*fi[pk-1]+(i-1)*fi[k])/m;
         }
      }
   }
}
```

Subroutine: (ntangle.cpp)

```
// ntangle.cpp (New program of triangle)
// Author: M. D. Choudhry
//
void NTANGLE(float s[],int m)
{
      int i,hm,lemda;

      hm=m/2;
      for(i=1;i<=hm;i++){
            s[i]=(float)i/hm;
            s[hm+i]=(float)(hm-(i-1))/hm;
      }
}
```

Subroutine: (gausscr.cpp)

```
// gausscr.cpp
// On Multiplying the outputs with 2*width, we get constant relative bandwidth
// the task of the multiplication be performed here inside this function for practical purposes.
//
// Author: Prof. J. M. Blackledge
// Developer: M. D. Choudhry
//
#include <math.h>

void GAUSSCR(float s[], int n, int width, int p)
{
        int i,nn,x;
        float pi,alpha;

        nn=2*width;
        pi = 4.0 * atan( 1.0 );
        alpha=width*width/4;
        for(i=1;i<n+1;i++){
             x=i-nn;
             s[i] = (0.5/pow(pi*alpha,0.5)) * exp(-((x-p)*(x-p)/(4.*alpha)));
        }
}
```

Subroutine: (wdmrrb.cpp)

```
// wdmrrb.cpp
// Wavelet decomposition with multiresolution relative bandwidth.
// Author: M. D. Choudhry
//
#include <stdlib.h>
#include <alloc.h>
#include <math.h>
#include "nrzozi.h"

void WDMRRB(float fi[], float fo[],int n,int j, int sgn)
{
        int i,a,k,r,dw,ha,width;
        float *w,*g,*zi;

        g=(float *)calloc(n+1, sizeof(float));
        zi=(float *)calloc(n+1, sizeof(float));
        w=(float *)calloc(n+1, sizeof(float));

        a=pow(2,j);
        dw=n/a;

        if(sgn==1){
              for(r=2;r<=6;r++){
                   width=pow(2,r);
                   GAUSSCR(g,n,width,2*r);
```

```
                for(i=1,k=1;i<=n;k=k+1,i=i+a){
                    zi[k]=g[i]*fi[i];
                    w[k]=w[k]+width*g[i];
                }
                for(i=1;i<=dw;i++){
                    fo[i]=fo[i]+width*zi[i];
                }
            }
            for(i=1;i<=dw;i++){
                fo[i]=fo[i]/w[i];
            }
        }
        if(sgn==-1){
            w[0]=0.;
            for(i=1;i<=a;i++) w[i]=(float)i;
            for(r=0;r<n;r++){
                for(i=1;i<=a;i++){
                    fo[r*a+i]=(w[a-i+1]*fi[a+r]+w[i-1]*fi[a+r+1])/a;
                }
            }
        }
        free (w);free (g); free (zi);
}
```

Subroutine: (bcgauss.cpp)

```
// bcgauss.cpp (Binary Coded Gaussian)
// Author: M. D. Choudhry
//
#include <alloc.h>
#include <math.h>

void BCGAUSS(int s[], int n, int width, int p)
{
        int i,nn,x;
        float *g,pi,alpha,ex:

        g=(float *)calloc(n+1, sizeof(float));
        nn=2*width;
        pi = 4.0 * atan( 1.0 );
        alpha=width*width/4;
        for(i=1;i<n+1;i++){
            x=i-nn;
            ex=exp(-(x-p)*(x-p)/(4.*alpha));
            g[i] = 2*width*(0.5/pow(pi*alpha,0.5))*ex;
        }
        for(i=0;i<n;i++){
            if(g[i]<.3737)s[i]=-1;
            else if(g[i]<.7474)s[i]=0;
            else if(g[i]<1.1284)s[i]=1;
        }
        free (g);
}
```